2016级数据结构第四次上机解题报告

A

这道题考察的是非常简单的队列的操作，也就是入队，出队，查询队首的操作。不合法的操作包括：队伍为空时进行出对，查询操作，队伍满时进行入队操作。

参考代码：

#include <iostream>

#define in std::cin

#define out std::cout

using namespace std;

struct MyQueue {

int head, tail, size, length;

int \*data;

MyQueue(int l) {

size = l;

head = tail = length = 0;

data = new int[size];

}

bool empty() {

return length == 0;

}

bool full() {

return length == size;

}

void offer(int x) {

data[tail] = x;

tail = (tail + 1) % size;

++length;

}

void poll() {

head = (head + 1) % size;

--length;

}

int peek() {

return data[head];

}

~MyQueue() {

delete data;

}

};

const int SIZE = 20;

int main() {

char cmd[8];

MyQueue q(SIZE);

while (in >> cmd) {

switch (cmd[1]) {

case 'f':

int val;

in >> val;

if (q.full()) {

out << "fool!" << "\n";

} else {

q.offer(val);

}

break;

case 'o':

if (q.empty()) {

out << "fool!" << "\n";

} else {

q.poll();

}

break;

case 'e':

if (q.empty()) {

out << "fool!" << "\n";

} else {

out << q.peek() << "\n";

}

break;

default:

break;

}

}

return 0;

}

B

中等题，需要进行一些推理，进行n-1次合并，每次将重量最小的两堆进行合并即可。

参考代码：  
#include<cstdio>

#include <iostream>

#include<algorithm>

using namespace std;

int main() {

int n;

scanf("%d", &n);

int a[n];

for (int i = 0; i < n; ++i) {

scanf("%d", &a[i]);

}

sort(a, a + n);

int ans = 0;

for (int i = 0; i < n - 1; ++i) {

ans += a[i] + a[i + 1];

a[i + 1] += a[i];

sort(a + i + 1, a + n);

}

printf("%d\n", ans);

return 0;

}

C

本题是一道很简单的递推题，唯一要注意的点是数据会爆long long，所以要用高精度。

参考代码：

#include <iostream>

#include <cstdlib>

#include <cstdio>

#include <string>

#include <algorithm>

#include <cmath>

#include <deque>

#include <vector>

#include <cstring>

using namespace std;

cnst int Base = 1000000000;

const int Capacity = 100;

typedef long long huge;

struct BigInt {

int Len;

int Data[Capacity];

BigInt() : Len(0) {}

BigInt(const BigInt &V) : Len(V.Len) { memcpy(Data, V.Data, Len \* sizeof\*Data); }

BigInt(int V) : Len(0) { for (; V>0; V /= Base) Data[Len++] = V%Base; }

BigInt &operator=(const BigInt &V) { Len = V.Len; memcpy(Data, V.Data, Len \* sizeof\*Data); return \*this; }

int &operator[] (int Index) { return Data[Index]; }

int operator[] (int Index) const { return Data[Index]; }

};

BigInt operator+(const BigInt &A, const BigInt &B) {

int i, Carry(0);

BigInt R;

for (i = 0; i<A.Len || i<B.Len || Carry>0; i++) {

if (i<A.Len) Carry += A[i];

if (i<B.Len) Carry += B[i];

R[i] = Carry%Base;

Carry /= Base;

}

R.Len = i;

return R;

}

ostream &operator<<(ostream &Out, const BigInt &V) {

int i;

Out << (V.Len == 0 ? 0 : V[V.Len - 1]);

for (i = V.Len - 2; i >= 0; i--) for (int j = Base / 10; j>0; j /= 10) Out << V[i] / j % 10;

return Out;

}

BigInt ans[300];

int main()

{

ans[0] = 1;

ans[1] = 1;

int i;

for (i = 2; i <= 250; i++)

{

ans[i] = ans[i - 2] + ans[i - 1] + ans[i - 2];

}

int n;

while (cin >> n)

{

n /= 3;

cout << ans[n] << endl;

}

return 0;

}

D

这道题属于一道中等题，对采取的算法并没有要求，只要按照题目的要求做出来即可。有一种方法是维护一个栈或者队列，从左向右扫描，如果找到一个比最左边的石头还要高的，就计算此时可以容纳的水量，也就是最左边石头的高度减去其他石头的高度。如果不比它高，就将这个石头的高度入栈。当全部扫完后，栈内有可能还有剩余，我们只需要把剩余的可容纳水量加上就行了。当然这道题还有很多其他的方法，就不一一介绍了。

参考代码：

#include <iostream>

#include <vector>

#include <fstream>

#define in std::cin

#define out std::cout

using namespace std;

int height[1000];

int stack[1000];

int main() {

int len;

while (in >> len) {

for (int i = 0; i < len; ++i) {

int val;

in >> val;

height[i] = val;

}

int top = -1, ret = 0;

for (int i = 0; i < len; i++) {

int x = height[i];

if (top == -1) {

stack[++top] = x;

} else {

int left = stack[0];

if (x > left) {

for (int j = 1; j <= top; j++) {

ret += left - stack[j];

}

stack[0] = x;

top = 0;

} else {

stack[++top] = x;

}

}

}

int cur = 0;

if (top > -1) {

cur = stack[top];

}

for (int i = top - 1; i > 0; i--) {

if (cur > stack[i]) {

ret += cur - stack[i];

} else {

cur = stack[i];

}

}

out << ret << "\n";

}

return 0;

}

E

首先容易证明，当盘子的个数为n时，移动的次数应等于2^n - 1。

一位美国学者发现一种出人意料的方法，只要轮流进行两步操作就可以了。

首先把三根柱子按顺序排成品字型，把所有的圆盘按从大到小的顺序放在柱子A上。

根据圆盘的数量确定柱子的排放顺序：若n为偶数，按顺时针方向依次摆放 A B C；

若n为奇数，按顺时针方向依次摆放 A C B。

（1）按顺时针方向把圆盘1从现在的柱子移动到下一根柱子，即当n为偶数时，若圆盘1在柱子A，则把它移动到B；若圆盘1在柱子B，则把它移动到C；若圆盘1在柱子C，则把它移动到A。

（2）接着，把另外两根柱子上可以移动的圆盘移动到新的柱子上。即把非空柱子上的圆盘移动到空柱子上，当两根柱子都非空时，移动较小的圆盘这一步没有明确规定移动哪个圆盘，你可能以为会有多种可能性，其实不然，可实施的行动是唯一的。

（3）反复进行（1）（2）操作，最后就能按规定完成汉诺塔的移动。

该算法的实现代码如下：

参考代码：

#include <iostream>

#include<time.h>

using namespace std;

//圆盘的个数最多为64

const int MAX = 64;

//用来表示每根柱子的信息

struct st {

int s[MAX]; //柱子上的圆盘存储情况

int top; //栈顶，用来最上面的圆盘

char name; //柱子的名字，可以是A，B，C中的一个

int Top()//取栈顶元素

{

return s[top];

}

int Pop()//出栈

{

return s[top--];

}

void Push(int x)//入栈

{

s[++top] = x;

}

};

long Pow(int x, int y); //计算x^y

void Creat(st ta[], int n); //给结构数组设置初值

void Hannuota(st ta[], long max); //移动汉诺塔的主要函数

int main(void) {

clock\_t start, finish;

int n;

cout << "请输入汉诺塔的阶数：";

cin >> n; //输入圆盘的个数

start = clock();

st ta[3]; //三根柱子的信息用结构数组存储

Creat(ta, n); //给结构数组设置初值

long max = Pow(2, n) - 1;//动的次数应等于2^n - 1

Hannuota(ta, max);//移动汉诺塔的主要函数

finish = clock();

printf("解决此 %d 阶汉诺塔所需的时间为：%.2f ms\n", n, (double) (finish - start));

system("pause");

return 0;

}

void Creat(st ta[], int n) {

ta[0].name = 'A';

ta[0].top = n - 1;

//把所有的圆盘按从大到小的顺序放在柱子A上

for (int i = 0; i < n; i++)

ta[0].s[i] = n - i;

//柱子B，C上开始没有没有圆盘

ta[1].top = ta[2].top = 0;

for (int i = 0; i < n; i++)

ta[1].s[i] = ta[2].s[i] = 0;

//若n为偶数，按顺时针方向依次摆放 A B C

if (n % 2 == 0) {

ta[1].name = 'B';

ta[2].name = 'C';

} else //若n为奇数，按顺时针方向依次摆放 A C B

{

ta[1].name = 'C';

ta[2].name = 'B';

}

}

long Pow(int x, int y) {

long sum = 1;

for (int i = 0; i < y; i++)

sum \*= x;

return sum;

}

void Hannuota(st ta[], long max) {

int k = 0; //累计移动的次数

int i = 0;

int ch;

while (k < max) {

//按顺时针方向把圆盘1从现在的柱子移动到下一根柱子

ch = ta[i % 3].Pop();

ta[(i + 1) % 3].Push(ch);

cout << ++k << ": " <<

"Move disk " << ch << " from " << ta[i % 3].name <<

" to " << ta[(i + 1) % 3].name << endl;

i++;

//把另外两根柱子上可以移动的圆盘移动到新的柱子上

if (k < max) {

//把非空柱子上的圆盘移动到空柱子上，当两根柱子都为空时，移动较小的圆盘

if (ta[(i + 1) % 3].Top() == 0 ||

ta[(i - 1) % 3].Top() > 0 &&

ta[(i + 1) % 3].Top() > ta[(i - 1) % 3].Top()) {

ch = ta[(i - 1) % 3].Pop();

ta[(i + 1) % 3].Push(ch);

cout << ++k << ": " << "Move disk"

<< ch << " from " << ta[(i - 1) % 3].name

<< " to " << ta[(i + 1) % 3].name << endl;

} else {

ch = ta[(i + 1) % 3].Pop();

ta[(i - 1) % 3].Push(ch);

cout << ++k << ": " << "Move disk"

<< ch << " from " << ta[(i + 1) % 3].name

<< " to " << ta[(i - 1) % 3].name << endl;

}

}

}

}

F

思路：

队列，先将重复数字压缩成1个，然后当每次一个元素进队列时，如果它和末位差超过1，就清空队列，如果差1，考虑和倒数第二位是否一样，一样就进队列，不一样就只保留原队列最后一位，做完处理后这个元素再进队列，中间出现过的最大的队列长度就是答案。

参考代码：

#include <cstdio>

#include <cmath>

#include <algorithm>

using namespace std;

long long a[100002];

long long q[100002][2];

int main()

{

long long n,cnt,e,add,ans;

while(scanf("%lld",&n)!=EOF)

{

for(long long i=0;i<n;i++)

scanf("%lld",&a[i]);

cnt=1;

e=0;

add=0;

ans=0;

for(long long i=1;i<=n;i++)

{

if(a[i]==a[i-1])

cnt++;

else

{

if(e==1&&abs(q[0][0]-a[i-1])>1)

{

e=0;

add=0;

}

else if(e>1&&abs(q[e-1][0]-a[i-1])>1)

{

e=0;

add=0;

}

else if(e>1&&q[e-2][0]!=a[i-1])

{

q[0][0]=q[e-1][0];

q[0][1]=q[e-1][1];

e=1;

add=q[e-1][1];

}

q[e][0]=a[i-1];

q[e][1]=cnt;

e++;

add+=cnt;

cnt=1;

if(ans<add)

ans=add;

}

}

printf("%lld\n",ans);

for(long long i=0;i<n;i++)

a[i]=0;

for(long long i=0;i<n;i++)

{

q[i][0]=0;

q[i][1]=0;

}

}

return 0;

}

G

表达式求值，裸题，书上一模一样，在此不做赘述，详情请看代码。

参考代码：

#include<bits/stdc++.h>

using namespace std;

struct cunchu

{

long long zhi;

bool mode;

};

struct slist

{

long long d;

slist \*link;

};

void init(slist \*&l)

{

l=new slist;

l->link=NULL;

}

void push(slist \*&l,long long x)

{

slist \*p=new slist;

p->d=x;

p->link=l->link;

l->link=p;

}

bool pop(slist \*&l,long long &x)

{

if (l->link==NULL)

return 0;

slist \*p=l->link;

l->link=p->link;

x=p->d;

delete p;

return 1;

}

bool isempty(slist \*&l) {return l->link==NULL;}

long long gettop(slist \*&l,long long &x)

{

if (l->link==NULL)

{

return 0;

}

x=l->link->d;

return 1;

}

int isp(char s)

{

if (s=='&')

return 7;

if (s=='#')

return 0;

if (s=='(')

return 1;

if (s=='\*'||s=='/'||s=='%')

return 5;

if (s=='+'||s=='-')

return 3;

if (s==')')

return 8;

}

int icp(char s)

{

if (s=='&')

return 6;

if (s=='#')

return 0;

if (s=='(')

return 8;

if (s=='\*'||s=='/'||s=='%')

return 4;

if (s=='+'||s=='-')

return 2;

if (s==')')

return 1;

}

void js(slist \*&l,int op)

{

long long left,right;

pop(l,right);

pop(l,left);

switch(op)

{

case '+':push(l,left+right);break;

case '-':push(l,left-right);break;

case '\*':push(l,left\*right);break;

case '/':push(l,left/right);break;

case '%':push(l,left%right);break;

case '&':

long long ttmp=pow(left,right);

push(l,ttmp);

break;

}

}

int main()

{

string s;

while (cin>>s)

{

if (s[0]=='-'||s[0]=='+')

s="0"+s;

for (int i=0;i<s.size();++i)

{

if (s[i]=='(')

{

if (s[i+1]=='-'||s[0]=='+')

{

i++;

s.insert(i,"0");

}

}

if (s[i]=='\*'&&s[i+1]=='\*')

{

s[i]='&';

string::iterator it=s.begin();

it+=i+1;

s.erase(it);

i--;

}

}

s+="#";

slist \*l;

init(l);

push(l,'#');

int wz=0;

vector<cunchu> hz;

long long x;

while (!isempty(l)&&wz!=s.size()+1)

{

if (isdigit(s[wz]))

{

int tmp=0;

while (isdigit(s[wz]))

{

tmp=tmp\*10+s[wz]-48;

wz++;

}

cunchu hhh;

hhh.zhi=tmp;

hhh.mode=1;

hz.push\_back(hhh);

}

else

{

gettop(l,x);

if (isp(x)<icp(s[wz]))

{

push(l,s[wz]);

wz++;

}

else if (isp(x)>icp(s[wz]))

{

long long op;

pop(l,op);

cunchu hhh;

hhh.zhi=op;

hhh.mode=0;

hz.push\_back(hhh);

}

else

{

long long op;

pop(l,op);

if (op=='(')

wz++;

}

}

}

slist \*open;

init(open);

long long ans;

wz=0;

for (int i=0;i<hz.size();++i)

{

if (!hz[i].mode)

js(open,hz[i].zhi);

else

push(open,hz[i].zhi);

}

pop(open,ans);

cout<<ans<<endl;

}

}