122492508@qq.com pq是不可战胜的

总分: 100 / 100

编程题 总分: 100 / 100

7-1 Conway's Conjecture 答案正确 得分: 20 / 20

John Horton Conway, a British mathematician active in recreational mathematics, proposed a conjecture in 2014: arrange the factors of any given number in ascending order, and pull the exponents down, we can get another number. Keep doing so we must end up at a prime number. For example:

$18 = 2 \times 3^2$

$232 = 2^3 \times 29$

$2329 = 17 \times 137$

17137 is a prime.

Now you are supposed to write a program to make one step verification of this conjecture. That is, for any given positive integer N, you must factorize it, and then test if the number obtained from its factors is a prime.

By the way, this conjecture has been proven false by James Davis, who has discovered a counter example: $135323853961879 = 13 \times 53^2 \times 3853 \times 96179$. Alas ...

Input Specification:

Each input file contains one test case which gives a positive integer N ($< 10^5$).

Output Specification:

For each case, first print in a line the number obtained from N's factors. The in the next line, print `Yes` if the above number is a prime, or `No` if not.

Sample Input 1:

```
2329
```

Sample Output 1:

```
17137
Yes
```

Sample Input 2:

```
124
```

Sample Output 2:

```
2231
No
```

Sample Input 3:

```
87516
```

Sample Output 3:

```
2232111317
No
```

```
/**
* 分析：数据类型需要使用long long，所以stoi变成stoll，坑点在于需要特判1
**/

#include <bits/stdc++.h>
using namespace std;
typedef long long LL;

struct factor {
        int x,cnt;
} fac[10];

bool isPrime(LL n) {
        if(n<=1)
                return false;
        LL sqr=(LL)sqrt(n*1.0);
        for(LL i=2; i<=sqr; i++) {
                if(n%i==0)
                        return false;
        }
        return true;
}

int main() {
        int n,num=0;
        scanf("%d",&n);
        if(n!=1) {
                int sqr=(int)(n*1.0);
                for(int i=2; i<=sqr; i++) {
                        if(isPrime(i)) {
                                if(n%i==0) {
                                        fac[num].x=i;
                                        fac[num].cnt=0;
                                        while(n%i==0) {
                                                fac[num].cnt++;
                                                n/=i;
                                        }
                                        num++;
                                }
                        }
                }
                if(n!=1) {
                        fac[num].x=n;
                        fac[num].cnt=1;
                        num++;
                }
                string s;
                for(int i=0; i<num; i++) {
                        //cout<<"fac[i].x="<<fac[i].x<<" fac[i].cnt="<<fac[i].cnt<<endl;
                        s+=to_string(fac[i].x);
                        if(fac[i].cnt!=1)
                                s+=to_string(fac[i].cnt);
                }
                cout<<s<<endl;
                LL temp=stoll(s);
                if(isPrime(temp))
                        printf("Yes\n");
                else
                        printf("No\n");
        } else {
                printf("1\nNo\n");
        }
        return 0;
}
```

测试点	结果	耗时	内存
0	答案正确	4 ms	384KB
1	答案正确	5 ms	384KB
2	答案正确	5 ms	424KB

测试点	结果	耗时	内存
3	答案正确	14 ms	424KB
4	答案正确	4 ms	360KB
5	答案正确	38 ms	384KB
6	答案正确	4 ms	508KB
7	答案正确	37 ms	384KB

7-2 Play with Linked List 答案正确 得分: 25 / 25

Given a singly linked list $L_1 \to L_2 \to \cdots \to L_{n-1} \to L_n$ and an integer $1 \le k < n$, you are supposed to rearrange the links to obtain a list like $L_k \to L_n \to L_{k-1} \to L_{n-1} \to \cdots$. For example, given L being 1→2→3→4→5→6 and $k = 4$, you must output 4→6→3→5→2→1.

Input Specification:

Each input file contains one test case. For each case, the first line contains the address of the first node, a positive N ($\le 10^5$) which is the total number of nodes, and an integer $1 \le k < n$ where n is the number of nodes in the linked list. The address of a node is a 5-digit nonnegative integer, and NULL is represented by `-1`.

Then N lines follow, each describes a node in the format:

```
Address Data Next
```

where `Address` is the position of the node, `Data` is a positive integer no more than 10^5, and `Next` is the position of the next node. It is guaranteed that there are at least two nodes on the list.

Output Specification:

For each case, output in order the resulting linked list. Each node occupies a line, and is printed in the same format as in the input.

Sample Input:

```
00100 6 4
00000 4 99999
00100 1 12309
68237 6 -1
33218 3 00000
99999 5 68237
12309 2 33218
```

Sample Output:

```
00000 4 68237
68237 6 33218
33218 3 99999
99999 5 12309
12309 2 00100
00100 1 -1
```

```
/**
* 分析：先根据题意获得新链表temp， 再根据temp获取顺序数组order
**/

#include <bits/stdc++.h>
using namespace std;
const int maxn = 100010;

int start,n,k;
int temp[maxn],order[maxn];

struct Node {
```

```
        int address,data,next;
        int order;
} node[maxn];

bool cmp(Node a,Node b) {
        return a.order<b.order;
}

// 原链表为 1 2 3 4 5 6    原链表结点在原链表中顺序为 1 2 3 4 5 6
// 新链表为 4 6 3 5 2 1    原链表结点在新链表中顺序为 6 5 3 1 4 2
// 原链表为 1 2 3 4 5 6 7  原链表结点在原链表中顺序为 1 2 3 4 5 6 7
// 新链表为 4 7 3 6 2 5 1  原链表结点在新链表中顺序为 7 5 3 1 6 4 2
void getorder(int cnt) { // k=4
        int i=1,p=k,q=cnt; //two pointer
        while(p>0&&q>k) {
                if(i%2!=0) {
                        temp[i]=p;
                        p--;
                } else if(i%2==0) {
                        temp[i]=q;
                        q--;
                }
                i++;
        }
        while(p>0) {
                temp[i]=p;
                p--;
                i++;
        }
        while(q>k) {
                temp[i]=q;
                q--;
                i++;
        }
        for(int i=1; i<=cnt; i++)
                order[temp[i]]=i;
}

int main() {
        for(int i=0; i<maxn; i++)
                node[i].order=maxn;
        int address,data,next;
        scanf("%d%d%d",&start,&n,&k);
        for(int i=0; i<n; i++) {
                scanf("%d",&address);
                node[address].address=address;
                scanf("%d%d",&node[address].data,&node[address].next);
        }
        int p=start,cnt=0;
        while(p!=-1) {
                node[p].order=cnt+1;
                p=node[p].next;
                cnt++;
        }
        getorder(cnt);
        p=start,cnt=0;
        while(p!=-1) {
                node[p].order=order[cnt+1];
                p=node[p].next;
                cnt++;
        }
        sort(node,node+maxn,cmp);
        for(int i=0; i<cnt; i++) {
                if(i!=cnt-1)
                        printf("%05d %d %05d\n",node[i].address,node[i].data,node[i+1].address);
                else
                        printf("%05d %d -1\n",node[i].address,node[i].data);
        }
        return 0;
}
```

测试点	结果	耗时	内存
0	答案正确	7 ms	1960KB
1	答案正确	7 ms	1920KB
2	答案正确	11 ms	1920KB
3	答案正确	9 ms	1920KB

测试点	结果	耗时	内存
4	答案正确	74 ms	4520KB

7-3 Unsuccessful Searches 答案正确 得分: 25 / 25

现有长度为 11 且初始为空的散列表 HT，散列函数是 $H(key) = key\%7$，采用线性探查（线性探测再散列）法解决冲突。将关键字序列 87,40,30,6,11,22,98,20 依次插入到 HT 后，HT 查找失败的平均查找长度是：(2分)

- A. 4
- B. 5.25
- C. 6
- D. 6.29

The above figure is a question from GRE-CS 2018. It states:

Given an initially empty hash table HT of size 11. The hash function is $H(key) = key\%7$, with linear probing used to resolve the collisions. Now hash the keys 87, 40, 30, 6, 11, 22, 98 and 20 one by one into HT. What is the average search time for **unsuccessful searches**?

The answer is 6.

Now you are supposed to write a program to solve this kind of problems.

Input Specification:

Each input file contains one test case. For each case, the first line gives 3 positive integers $TSize$ ($\le 10^3$, the table size), M ($\le TSize$, the divisor in the hash function), and N ($\le TSize$, the number of integers to be inserted). Then N non-negative integers ($\le 10^4$) are given in the next line, separated by spaces.

Output Specification:

Print in a line the average search time for unsuccessful searches, after hashing the N integers into the table. The answer must be accurate up to 1 decimal place.

Sample Input 1:

```
11 7 8
87 40 30 6 11 22 98 20
```

Sample Output 1:

```
6.0
```

Sample Input 2:

```
3 3 3
81 2 5
```

Sample Output 2:

```
4.0
```

Note: In sample 2, the last test of the original position counts as well.

```
/**
* 分析：注意ASL失败计算是计算mod即可
**/

#include <bits/stdc++.h>
using namespace std;
const int maxn = 1010;
```

```
int h[maxn];

int main() {
	fill(h,h+maxn,-1);
	int Tsize,m,n,key;
	double cnt=0;
	scanf("%d%d%d",&Tsize,&m,&n);
	for(int i=0; i<n; i++) {
		scanf("%d",&key);
		int hkey=key%m;
		for(int k=0; k<Tsize; k++) {
			int pos=(hkey+k)%Tsize;
			if(h[pos]==-1) {
				h[pos]=key;
				break;
			}
		}
	}
	for(int i=0; i<m; i++) {
		for(int k=0; k<=Tsize; k++) {
			cnt++;
			int pos=(i+k)%Tsize;
			if(h[pos]==-1)
				break;
		}
	}
	printf("%.1f\n",cnt/m);
	return 0;
}
```

测试点	结果	耗时	内存
0	答案正确	3 ms	416KB
1	答案正确	2 ms	416KB
2	答案正确	3 ms	356KB
3	答案正确	4 ms	416KB
4	答案正确	5 ms	384KB

7-4 Ambulance Dispatch 答案正确 得分: 30 / 30

Given the map of a city, with all the ambulance dispatch centers (救护车派遣中心) and all the pick-up spots marked. You are supposed to write a program to process the emergency calls. It is assumed that the callers are waiting at some pick-up spot. You must inform the nearest (that is, to take the minimum time to reach the spot) dispatch center if that center has at least one ambulance available. Note: a center without any ambulance must not be considered.

In case your options are not unique, inform the one with the largest number of ambulances available. If there is still a tie, choose the one that can pass the least number of streets to reach the spot, which is guaranteed to be unique.

Input Specification:

Each input file contains one test case. For each case, the first line contains two positive integers N_s ($\leq 10^3$) and N_a (≤ 10), which are the total number of pick-up spots and the number of ambulance dispatch centers, respectively. Hence the pick-up spots are numbered from 1 to N_s, and the ambulance dispatch centers are numbered from $A-1$ to $A-N_a$.

The next line gives N_a non-negative integers, where the i-th integer is the number of available ambulances at the i-th center. All the integers are no larger than 100.

In the next line a positive number M ($\leq 10^4$) is given as the number of streets connecting the spots and the centers. Then M lines follow, each describes a street by giving the indices of the spots or centers at the two ends, followed by the time taken to pass this street, which is a positive integer no larger than 100.

Finally the number of emergency calls, K, is given as a positive integer no larger than 10^3, followed by K indices of pick-up spots.

All the inputs in a line are separated by a space.

Output Specification:

For each of the K calls, first print in a line the path from the center that must send an ambulance to the calling spot. All the nodes must be separated by exactly one space and there must be no extra space at the beginning or the end of the line. Then print the

minimum time taken to reach the spot in the next line. It is assumed that the center will send an ambulance after each call. If no ambulance is available, just print `All Busy` in a line. It is guaranteed that all the spots are connected to all the centers.

Sample Input:

```
7 3
3 2 2
16
A-1 2 4
A-1 3 2
3 A-2 1
4 A-3 1
A-1 4 3
6 7 1
1 7 3
1 3 3
3 4 1
6 A-3 5
6 5 2
5 7 1
A-2 7 5
A-2 1 1
3 5 1
5 A-3 2
8
6 7 5 4 6 4 3 2
```

Sample Output:

```
A-3 5 6
4
A-2 3 5 7
3
A-3 5
2
A-2 3 4
2
A-1 3 5 6
5
A-1 4
3
A-1 3
2
All Busy
```

```
#include<stdio.h>
#include<queue>
#include<string.h>
#include<algorithm>
using namespace std;
const int maxn = 1020, inf = 1e9;
char c1[10], c2[10];
int ns, na;
int stoid(char c[]) {
        int ret;
        if (c[0] == 'A') {
                sscanf(c + 2, "%d", &ret);
                ret += ns;
        }
        else sscanf(c, "%d", &ret);
        return ret;
}
struct edge {
        int to, time;
};
```

```
vector<edge> g[maxn];
int d[maxn], pre[maxn], num[maxn], anum[maxn];
bool vis[maxn];
struct node {
	int v, d;
};
struct cmp {
	bool operator()(const node &n1, const node& n2) {
		return n1.d > n2.d;
	}
};
void Dijkstra(int s) {
	fill(d, d + maxn, inf);
	memset(vis, false, sizeof(vis));
	memset(pre, -1, sizeof(pre));
	memset(num, 0, sizeof(num));
	priority_queue<node, vector<node>, cmp> q;
	d[s] = 0;
	num[s] = 0;
	q.push(node{ s,0 });
	while (!q.empty()) {
		int u = q.top().v;
		q.pop();
		if (vis[u] == true) continue;
		vis[u] = true;
		for (int i = 0; i < g[u].size(); i++) {
			int v = g[u][i].to;
			if (vis[v] == false) {
				int time = g[u][i].time;
				if (d[u] + time < d[v]) {
					d[v] = d[u] + time;
					num[v] = num[u] + 1;
					pre[v] = u;
					q.push(node{ v,d[v] });
				}
				else if (d[u] + time == d[v]&& num[u] + 1 < num[v]) {
					num[v] = num[u] + 1;
					pre[v] = u;
					q.push(node{ v,d[v] });
				}
			}
		}
	}
}
int main() {
	scanf("%d%d", &ns, &na);
	for (int i = 1; i <= na; i++) {
		scanf("%d", &anum[i + ns]);
	}
	int m; scanf("%d", &m);
	for (int i = 0; i < m; i++) {
		int time, id1, id2;
		scanf("%s%s%d", c1, c2, &time);
		id1 = stoid(c1);
		id2 = stoid(c2);
		g[id1].push_back(edge{ id2,time });
		g[id2].push_back(edge{ id1,time });
	}
	int q; scanf("%d", &q);
	for (int i = 0; i < q; i++) {
		int ed;
		scanf("%d", &ed);
		Dijkstra(ed);
		int pos = -1, mintime = inf, minnum, maxa;
		for (int i = ns + 1; i <= ns + na; i++) {
			if (anum[i] == 0) continue;
			if (d[i] < mintime) {
				pos = i;
				mintime = d[i];
				minnum = num[i];
				maxa = anum[i];
			}
			else if (d[i] == mintime && anum[i] > maxa) {
				pos = i;
				minnum = num[i];
				maxa = anum[i];
			}
			else if (d[i] == mintime && anum[i] == maxa && num[i] < minnum) {
				pos = i;
				minnum = num[i];
```

```
                        }
                }
                if (pos != -1) {
                        printf("A-%d", pos - ns);
                        int p = pre[pos];
                        while (p != ed) {
                                if (p > ns) printf(" A-%d", p - ns);
                                else printf(" %d", p);
                                p = pre[p];
                        }
                        printf(" %d\n", ed);
                        printf("%d\n", d[pos]);
                        anum[pos]--;
                }
                else printf("All Busy\n");
        }
}
```

测试点	结果	耗时	内存
0	答案正确	2 ms	512KB
1	答案正确	2 ms	228KB
2	答案正确	2 ms	384KB
3	答案正确	2 ms	384KB
4	答案正确	317 ms	672KB