算法题汇总

1004 C语言&数据结构复习

A : log2(N)

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

给你一个数 N，请你找出最大的的整数 k，满足 $2^k \leq N$

Input

$1 \leq N \leq 10^{19}$

Output

输出 k 并换行

Sample Input

```
6
```

Sample Output

```
2
```

Hint

注意精度问题

```
/*
A - log2(N)
*/
#include <bits/stdc++.h>
using namespace std;

using ULL = unsigned long long;

int main() {
    ULL n;
    cin >> n;
    int k = 0;
    for ( ULL x = 1; x <= n; x <<= 1, k++ )
        ;
    cout << k - 1 << endl;
    return 0;
}

/*
```

```
input:
6

output:
2
*/
```

B : 如何溜的最快

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

给你一个终点 (x, y)，从 $(0, 0)$ 出发，一步最长为 R，可以走向任意方向，请你输出到达终点所需的最短步数。

Input

第一行输入 r, x, y

$1 \leq R \leq 10^5$

$0 \leq x, y \leq 10^5$

$(x, y) \neq (0, 0)$

所有输入皆为int型

Output

输出结果并换行

Sample Input

```
3 4 4
```

Sample Output

```
2
```

```cpp
/*
B - 如何溜的最快
*/
#include <bits/stdc++.h>
using namespace std;

int main() {
    int x, y, R;
    while ( cin >> x >> y >> R )
        cout << int( ceil( sqrt( x * x + y * y ) / R ) ) << endl;
    return 0;
}

/*
input:
```

```
3 4 4

output:
2
*/
```

C : 200和整数对之间的情缘

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

给你一个有 N 个整型数字的序列 A，整数对 (i, j) 满足以下条件，请问能找到多少这样的整数对：

- $1 \leq i, j \leq N$
- $A_i - A_j$ 是200的整数倍

Input

第一行输入 N，第二行输入 N 个整数分别 $A_1, A_2, \cdots, A_N$

- 所有输入均为整型(int)
- $2 \leq N \leq 2 \times 10^5$
- $1 \leq A_i \leq 10^9$

Output

输出答案并换行

Sample Input

```
6
123 223 123 523 200 2000
```

Sample Output

```
4
```

```
/*
C - 200和整数对之间的情缘
*/
#include <bits/stdc++.h>
using namespace std;

const int MAX = 200;

int main() {
    long long N;
    while ( cin >> N ) {
        vector<int> array( MAX, 0 );
        int num;
        for ( int i = 0; i < N; i++ ) {
            cin >> num;
            ++array[ num % MAX ];
```

```
        }

        long long ans = 0;
        for ( int i = 0; i < MAX; i++ ) {
            ans += array[ i ] * ( array[ i ] - 1 );
        }
        cout << ( ans >>= 1 ) << endl;
    }
    return 0;
}

/*
input:
6
123 223 123 523 200 2000

output:
4
*/
```

D : 两面包夹芝士

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

给你两个长度为 N 的整型(int)数组 $A = (A_1, A_2, \cdots, A_N)$ 和 $B = (B_1, B_2, \cdots, B_N)$

请你找出有多少整型(int)数字 x，满足 $A_i \leq x \leq B_i$，其中 $1 \leq i \leq N$

Input

第一行输入N，第二行输入 N 个数字 A_i，第三行输入N个数字 B_i

- $1 \leq N \leq 100$
- $1 \leq A_i \leq B_i \leq 1000$
- 所有输入均为int

Output

输出有多少个符合题意的 x，然后换行

Sample Input

```
3
3 2 5
6 9 8
```

Sample Output

```
2
```

```
/*
D - 两面包夹芝士
```

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

int main() {
    int N, num;
    int maxA = -1, minB = 1000000;

    cin >> N;

    for ( int i = 0; i < N; i++ ) {
        cin >> num;
        maxA = ( num > maxA ) ? num : maxA;
    }

    for ( int i = 0; i < N; i++ ) {
        cin >> num;
        minB = ( num < minB ) ? num : minB;
    }

    cout << ( ( minB >= maxA ) ? ( minB - maxA + 1 ) : 0 ) << endl;

    return 0;
}

/*
input:
3
3 2 5
6 9 8

output:
2
*/
```

E : 构造回文串

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

给你一个整型(int)数 N，可以将其看成字符串，是否有可能在 N 前加入若干个0(也可以不加)，使得该字符串为回文串。

Input

数字 N ($0 \leq N \leq 10^9$)

Output

若可以则输出Yes，不可以则输出No，输出结束要换行。

Sample Input

```
1210
```

Sample Output

```
Yes
```

```cpp
/*
E - 构造回文串
*/
#include <bits/stdc++.h>
using namespace std;

int main() {
    // long long N = 1234210000;
    long long N;
    cin >> N;

    while ( N % 10 == 0 ) {
        N /= 10;
    }

    string numStr = to_string( N );

    bool break_flag = false;
    for ( int cursor1 = 0, cursor2 = numStr.size() - 1;
          cursor1 <= cursor2 && cursor1 != cursor2; ++cursor1, --cursor2 ) {
        // cout << "num[cursor1]" << numStr.at(cursor1) << ", "
        //      << "num[cursor2] = " << numStr.at(cursor2) << endl;
        if ( numStr.at( cursor1 ) != numStr.at( cursor2 ) ) {
            break_flag = true;
            break;
        }
    }

    cout << ( !break_flag ? "Yes" : "No" ) << endl;
    return 0;
}

/*
input:
1210

output:
Yes
*/
```

F：模拟计算器

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

给出 n 个数，和 $n-1$个运算符（只含有加减乘号，不含除号，按顺序填入 n 个数之间），要求输出该式的答案。

Input

单组数据。

第一行为 n（$2 \leq n \leq 100$），且 n 为整数。

第二行为 n 个数，每个数 a 保证 $1 \leq a \leq 10$，且为整数。

第三行为 $n-1$个运算符，运算符之间由空格隔开。

Output

如题。

Sample Input

```
4
1 2 3 4
+ * -
```

Sample Output

```
3
```

Hint

1 + 2 * 3 - 4 = 3

```cpp
/*
F - 模拟计算器
*/
#include <bits/stdc++.h>
using namespace std;

long long calculator( queue<long long> numq, long long N ) {
    stack<long long> nums;
    string op;

    nums.push( numq.front() );
    numq.pop();
    for ( int i = 1; i < N; i++ ) {
        long long num = numq.front();
        numq.pop();

        cin >> op;
        if ( op.at( 0 ) == '+' ) {
            nums.push( num );
        } else if ( op.at( 0 ) == '-' ) {
            nums.push( -num );
        } else if ( op.at( 0 ) == '*' ) {
```

```
            long long top = nums.top();
            nums.pop();
            nums.push( num * top );
        }
    }

    long long result = 0, size = nums.size();
    for ( int i = 0; i < size; i++ ) {
        result += nums.top();
        nums.pop();
    }
    return result;
}

int main() {
    long long N, num;
    cin >> N;

    queue<long long> numq;

    for ( int i = 0; i < N; i++ ) {
        cin >> num;
        numq.push( num );
    }

    long long result = calculator( numq, N );
    cout << result << endl;
    return 0;
}

/*
input:
4
1 2 3 4
+ * -

output:
3
*/
```

G：排队援救

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

突发重大灾难，n 个人陷入困境，有一个救援点需要人们排队准备接受救援，假设每个人有一个名望值，进队规则如下：

- 第一个人直接进队；
- 当队里已经有人时，新来的人发现队尾的人的名望值比自己大或者相等，他会选择离开去其他救援点；
- 队伍最多5人，如果一个人要进队时，发现队伍已满，而且他的名望值比队尾的人大，他会选择把队首的人挤掉而继续排在队尾。

问最后得到救援的人分别是谁。

Input

单组数据。

第一行为 n（$1 \leq n \leq 100$），n 为正整数。

第二行为 n 个人的名望值，第 i 个去排队的人的名望值为 a_i（$1 \leq a_i \leq 2^{32}-1$），且为正整数。

Output

按顺序输出最后得到救援的人的号码，一个人号码是多少即为他是第几个去排队的。

Sample Input

```
6
1 3 5 7 9 11
```

Sample Output

```
2 3 4 5 6
```

```cpp
/*
G - 排队援救
*/
#include <bits/stdc++.h>
using namespace std;

const int MAX = 5;

int main() {
    int n, cnt = 0;
    long long value;
    queue<long long> valueQ;
    queue<int> seqQ;

    cin >> n;

    for ( int i = 1; i <= n; ++i ) {
        cin >> value;

        if ( cnt > 0 && value <= valueQ.back() ) continue;

        if ( cnt < MAX ) {
            valueQ.push( value );
            seqQ.push( i );
            // cout << "push: " << i << "[value=" << value << "] " << endl;
            ++cnt;
        } else if ( cnt == MAX ) {
            // cout << "pop: " << seqQ.front() << "[value=" << value << "] ";
            valueQ.pop();
            seqQ.pop();

            valueQ.push( value );
            seqQ.push( i );
            // cout << ", push: " << i << "[value=" << value << "] " << endl;
        }
```

```
    }

    while ( seqQ.size() > 0 ) {
        cout << seqQ.front() << " ";
        seqQ.pop();
    }
    cout << endl;
    return 0;
}

/*
input:
6
1 3 5 7 9 11

output:
2 3 4 5 6
*/
```

H : 括号匹配

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

给一个长度不大于100的字符串，问该字符串里出现的括号是否合法。

字符串只会出现三种括号，“[]”，“()”，“{}”，且形如“({[]})”为合法，形如“[(])”为不合法。

Input

单组数据。

长度不大于100的字符串，字符串中不存在空格。

Output

如果该字符串中出现的所有括号都合法，输出“Yes”，否则输出“No”。

Sample Input

```
(a+b)!=c
```

Sample Output

```
Yes
```

```
/*
H - 括号匹配
*/
#include <bits/stdc++.h>
using namespace std;

bool isLeftBracket( char ch ) {
```

```
    bool isleft = false;
    switch ( ch ) {
    case '(':
    case '[':
    case '{':
        isleft = true;
        break;
    default:
        break;
    }
    return isleft;
}

bool isRightBracket( char ch ) {
    bool isright = false;
    switch ( ch ) {
    case ')':
    case ']':
    case '}':
        isright = true;
        break;
    default:
        break;
    }
    return isright;
}

bool isMatchedWithTop( char ch, stack<char> charStack ) {
    bool isMatched = false;
    if ( charStack.empty() ) return isMatched;

    char top = charStack.top();
    switch ( ch ) {
    case '(':
        if ( top == ')' ) isMatched = true;
        break;
    case '[':
        if ( top == ']' ) isMatched = true;
        break;
    case '{':
        if ( top == '}' ) isMatched = true;
        break;
    case ')':
        if ( top == '(' ) isMatched = true;
        break;
    case ']':
        if ( top == '[' ) isMatched = true;
        break;
    case '}':
        if ( top == '{' ) isMatched = true;
        break;
    default:
        break;
    }
    return isMatched;
}

int main() {
```

```
    stack<char> brackets;
    string str;
    bool islegal = false, isbreaked = false;

    getline( cin, str );
    for ( auto it = str.cbegin(); it != str.end(); ++it ) {
        if ( isLeftBracket( *it ) ) {
            brackets.push( *it );
            // cout << "pushed: " << brackets.top() << endl;
        } else if ( isRightBracket( *it ) ) {
            if ( !isMatchedWithTop( *it, brackets ) ) {
                // cout << "breaked with: " << *it << endl;
                isbreaked = true;
                break;
            } else {
                // cout << "poped: " << brackets.top() << endl;
                brackets.pop();
            }
        }
    }

    if ( !isbreaked && brackets.empty() ) islegal = true;
    cout << ( islegal ? "Yes" : "No" ) << endl;
    return 0;
}

/*
input:
(a+b)!=c

output:
Yes
*/
```

I : 异或最大值升级版

Time Limit: `2 Sec`, Memory Limit: `128 Mb`

Description

给 n 个数 `a[1] ~ a[n]`，求 `(a[i] + a[j]) ⊕ a[k]` 的最大值，其中 `i, j, k` 为互不相同的序号，“⊕”表示按位异或。

Input

多组数据，每组数据第一行一个 n，第二行 n 个正整数 `a[i]`。

其中 `3 <= n <= 2000`， `0 <= a[i] <= 10^9`。

Output

每组数据输出最大的结果。

Sample Input

```
3
3 1 2
5
1 7 6 8 9
```

Sample Output

```
6
24
```

```cpp
/*
I - 异或最大值升级版

reference:
https://dwz.date/ffRD
*/

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

// #define DEBUG

const int HIGH_BIT = 30;
const int MAXNUM = 100000;

struct Digit {
    int pathcnt = 0;
    int next[ 2 ] = { -1, -1 };

    void Init() {
        next[ 0 ] = next[ 1 ] = -1;
        pathcnt = 0;
    }
};

class BinaryDigits {
  private:
    vector<Digit> binums = vector<Digit>( MAXNUM );
    int nextpos = 1;

  public:
    void reset() {
        binums[ 0 ].Init();
        nextpos = 1;
    }

    void insert( int num ) {
        int cursor = 0;
        for ( int k = HIGH_BIT; k >= 0; --k ) {
            int bit = num >> k & 1;
            if ( binums[ cursor ].next[ bit ] == -1 ) {
                binums[ nextpos ].Init();
                binums[ cursor ].next[ bit ] = nextpos++;
```

```
            }
            ++( binums[ cursor ].pathcnt );
            cursor = binums[ cursor ].next[ bit ];
        }
        ++( binums[ cursor ].pathcnt );
    }

    void remove( int num ) {
        int cursor = 0;
        for ( int k = HIGH_BIT; k >= 0; --k ) {
            int bit = num >> k & 1;
            --( binums[ cursor ].pathcnt );
            cursor = binums[ cursor ].next[ bit ];
        }
        --( binums[ cursor ].pathcnt );
    }

    int findMaxXor( int num ) {
        int cursor = 0, x = 0;
        for ( int k = HIGH_BIT; k >= 0; --k ) {
            int bit = ( num >> k & 1 ) ^ 1;
            bool isone = true;

            int pos_to_check = binums[ cursor ].next[ bit ];
            if ( pos_to_check == -1 || !binums[ pos_to_check ].pathcnt )
                bit ^= 1, isone = false;

            if ( binums[ cursor ].pathcnt == 0 ||
                 binums[ cursor ].next[ bit ] == -1 )
                return 0;

            cursor = binums[ cursor ].next[ bit ];
            x = isone ? 2 * x + 1 : 2 * x;
        }
        return binums[ cursor ].pathcnt > 0 ? x : 0;
    }

#ifdef DEBUG
    void display_binums() {
        for ( int k = HIGH_BIT * 2; k >= 0; --k ) {
            cout << "[k = " << k << "]: ";
            cout << "cnt=" << binums[ k ].pathcnt << " ";
            cout << "[0]:" << binums[ k ].next[ 0 ] << ", ";
            cout << "[1]:" << binums[ k ].next[ 1 ] << endl;
        }
        cout << endl;
    }
#endif
};

int main() {
    BinaryDigits bidigits;

    int n;
    while ( cin >> n ) {
        bidigits.reset();

        vector<int> nums( n );
```

```
        for ( int i = 0; i < n; ++i ) {
            cin >> nums[ i ];
            bidigits.insert( nums[ i ] );
        }

#ifdef DEBUG
        cout << ">> after insert:" << endl;
        bidigits.display_binums();
#endif

        int ans = 0;
        for ( int i = 0; i < n; ++i ) {
            bidigits.remove( nums[ i ] );
            for ( int j = i + 1; j < n; ++j ) {
                bidigits.remove( nums[ j ] );
                ans = std::max( ans,
                                bidigits.findMaxXor( nums[ i ] + nums[ j ] ) );
                bidigits.insert( nums[ j ] );
            }
            bidigits.insert( nums[ i ] );
        }
        cout << ans << endl;
    }
    return 0;
}

/*
input:
3
3 1 2
5
1 7 6 8 9

output:
6
24
*/
```

1015 递推与分治基础

A : 禽兽的传染病

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

禽兽患传染病了。一个禽兽会每轮传染 x 个禽兽。试问 n 轮传染后有多少禽兽被传染?

Input

两个非负整数 x 和 n。

Output

一个整数，即被传染的禽兽数。

Sample Input

```
10 2
```

Sample Output

```
121
```

```cpp
/*
A - 禽兽的传染病
*/
#include <bits/stdc++.h>
using namespace std;

int main() {
    long long x, n;
    cin >> x >> n;

    long long result = 1;
    for ( int i = 0; i < n; ++i )
        result *= ( 1 + x );
    cout << result << endl;
    return 0;
}

/*
input:
10 2

output:
121
*/
```

B : 台阶问题

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

有 N 级的台阶，你一开始在底部，每次可以向上迈最多 K 级台阶（最少 1 级），问到达第 N 级台阶有多少种不同方式。

Input

两个正整数 N, K。$1 \leq N \leq 10^6, 1 \leq K \leq 20$

Output

一个正整数，为不同方式数，由于答案可能很大，你需要输出 ans mod 100003 后的结果。

Sample Input

```
5 2
```

Sample Output

```
8
```

```cpp
/*
B - 台阶问题

reference:
https://www.geek-share.com/detail/2790040145.html
*/

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

using ULL = unsigned long long;

const int MAXN = 100000 + 10;
const int num_to_mod = 100003;

int main() {
    int N, k;
    cin >> N >> k;

    vector<ULL> nums( MAXN );

    nums[ 0 ] = 1;

    for ( int i = 1; i <= N; ++i ) {
        for ( int j = 1; j <= k; ++j ) {
            if ( i >= j ) {
                nums[ i ] += nums[ i - j ];
                nums[ i ] %= num_to_mod;
            }
        }
    }
    cout << nums[ N ] << endl;
    return 0;
}

/*
input:
5 2

output:
8
*/
```

C : 数的计算

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

我们要求找出具有下列性质数的个数(包含输入的正整数 n)。

先输入一个正整数 n ($n \leq 1000$),然后对此正整数按照如下方法进行处理:

1. 不作任何处理;
2. 在它的左边加上一个正整数,但该正整数不能超过原数的一半;
3. 加上数后,继续按此规则进行处理,直到不能再加正整数为止。

Input

1 个正整数 n ($n \leq 1000$)

Output

1 个整数，表示具有该性质数的个数。

Sample Input

```
6
```

Sample Output

```
6
```

```cpp
/*
C - 数的计算

reference:
https://www.shuzhiduo.com/A/n2d9M7QQdD/
*/

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

int main() {
    int n;
    cin >> n;

    int nums[ 1010 ];

    nums[ 0 ] = nums[ 1 ] = 1;
    for ( int i = 2; i <= n; ++i ) {
        nums[ i ] = ( i % 2 ) ? nums[ i - 1 ] : nums[ i - 1 ] + nums[ i / 2 ];
    }

    cout << nums[ n ] << endl;
    return 0;
}
```

```
/*
input:
6

output:
6
*/
```

D : 数的划分

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

将整数 n 分成 k 份，且每份不能为空，任意两个方案不相同（不考虑顺序）。

例如：$n=7,k=3$，下面三种分法被认为是相同的。

1 , 1 , 5

1 , 5 , 1

5 , 1 , 1

问有多少种不同的分法。

Input

n,k $(6<n\leq 200,\ 2\leq k\leq 6)$

Output

1 个整数，即不同的分法。

Sample Input

```
7 3
```

Sample Output

```
4
```

Hint

对于样例的输入输出

四种分法为:

1 , 1 , 5

1 , 2 , 4

1 , 3 , 3

2 , 2 , 3

```
```

```cpp
/*
D - 数的划分

reference:
https://www.nowcoder.com/questionTerminal/3773e51c48ec4727939cc85a8bc4c60d?
answerType=1&f=discussion
*/

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

int ans = 0;

void recurse( int total, int num, int cnt ) {
    if ( cnt == 0 ) {
        if ( total == 0 ) ++ans;
        return;
    }

    for ( int i = num; i <= total; ++i )
        recurse( total - i, i, cnt - 1 );
}

int main() {
    int n, k;
    cin >> n >> k;

    recurse( n, 1, k );
    cout << ans << endl;
    return 0;
}

/*
input:
7 3

output:
4
*/
```

E : 绝对值最小的和

Time Limit: `2 Sec`, Memory Limit: `128 Mb`

Description

给出一串由 *N* 个整数组成的数列 $a_1, a_2, a_3, \cdots, a_N$ 求问数列中哪两个不同元素的和的绝对值最小。

Input

输入共一组数据，共2行

第一行一个数字 N，表示数列中元素的个数

第二行共 N 个数字，两两之间用空格分开，表示

$a_1, a_2, a_3, \cdots, a_N$

输入保证 $2 \leq N \leq 1 \times 10^3$，$-1 \times 10^8 \leq a_i \leq 1 \times 10^8$

Output

输出共1行，两个用空格隔开的数字 i 和 j，表示 $a_i + a_j$的绝对值最小。

若两对元素的之和的绝对值相同，且都比其它对元素之和的绝对值小，则取编号较小元素的编号较小的一对；若编号较小元素相同，则取编号较大元素的编号较小的一位。

例如，数列：-1, 1, 1, -1 中，有 $|a_1 + a_2| = |a_1 + a_3| = |a_2 + a_4| = |a_3 + a_4| = 0$

，它们的绝对值相等且最小。但应输出`1 2`

Sample Input

```
5
5 4 3 2 1
```

Sample Output

```
4 5
```

```cpp
/*
E - 绝对值最小的和
*/
#include <bits/stdc++.h>
using namespace std;

// #define DEBUG

int main() {
    int N;
    cin >> N;

    vector<int> nums( N + 1, 0 );

    for ( int i = 1; i <= N; ++i )
        cin >> nums[ i ];

    vector<vector<int>> min_results( N + 1 );
    min_results[ 0 ] = { 1, 2, abs( nums[ 1 ] + nums[ 2 ] ) };

    for ( int i = 1; i < N; ++i ) {
        min_results[ i ] = { i, i + 1, abs( nums[ i ] + nums[ i + 1 ] ) };

        for ( int j = i + 1; j <= N; ++j ) {
            if ( abs( nums[ i ] + nums[ j ] ) < min_results[ i ][ 2 ] )
                min_results[ i ] = { i, j, abs( nums[ i ] + nums[ j ] ) };
        }

        if ( min_results[ i ][ 2 ] < min_results[ 0 ][ 2 ] )
            min_results[ 0 ] = min_results[ i ];
    }

#ifdef DEBUG
    for ( int i = 1; i < N; ++i )
```

```
        cout << min_results[ i ][ 0 ] << " " << min_results[ i ][ 1 ] << " "
             << min_results[ i ][ 2 ] << endl;
    cout << endl;
#endif

    cout << min_results[ 0 ][ 0 ] << " " << min_results[ 0 ][ 1 ] << endl;
    return 0;
}

/*
input:
5
5 4 3 2 1

output:
4 5
*/
```

F：区间和

Time Limit: `2 Sec`， Memory Limit: `128 Mb`

Description

给出一串由N个自然数组成的数列 $a_1, a_2, a_3, \ldots, a_N$，之后给出M次询问，每次询问包含两个数字L和R。对于每次询问，求出L和R之间所有数（包括L和R本身）之和。

Input

输入仅有一组数据，共 *M* + 2行。

第一行有两个用空格隔开的自然数 *N* 和 *M*，表示数列中共有*N*个数字，一共有 *M* 次询问。保证 $1 \leq N, M \leq 1 \times 10^6$ 第二行有 *N* 个用空格隔开的自然数，$a_1, a_2, a_3, \ldots, a_N$对于所有的*i*，都有 $1 \leq a_i \leq 1 \times 10^6$

第三行到第*M*+2行，每行有两个自然数 *L* 和 *R*，表示要问数列中a_L到a_R之间的所有数之和（包括a_L和a_R）。保证 1 ≤ *L* ≤ *R* ≤ *N*

Output

输出共 *M*行，每行1个自然数，即数列中a_L到a_R之间的所有数之和（包括a_L和a_R）

Sample Input

```
5 8
1 2 3 4 5
1 3
2 5
2 3
1 1
5 5
1 5
3 4
4 5
```

Sample Output

```
6
14
5
1
5
15
7
9
```

```cpp
/*
F - 区间和
*/
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 1000000 + 10;

long long nums[ MAXN ];
long long sums[ MAXN ];

int main() {
    long long N, M;
    scanf( "%lld %lld", &N, &M );

    nums[ 0 ] = sums[ 0 ] = 0;
    for ( long long i = 1; i <= N; ++i )
        scanf( "%lld", &nums[ i ] );

    sums[ 1 ] = nums[ 1 ];

    for ( long long i = 2; i <= N; ++i )
        sums[ i ] = sums[ i - 1 ] + nums[ i ];

    for ( long long i = 0; i < M; ++i ) {
        long long l, r;
        scanf( "%lld %lld", &l, &r );
        printf( "%lld\n", sums[ r ] - sums[ l - 1 ] );
    }

    return 0;
}

/*
input:
5 8
1 2 3 4 5
1 3
2 5
2 3
1 1
5 5
1 5
3 4
4 5
```

```
output:
6
14
5
1
5
15
7
9
*/
```

G : 查找[二分]

Time Limit: `2 Sec`, Memory Limit: `128 Mb`

Description

给出一串由N个自然数组成的数列 $a_1, a_2, a_3, \ldots, a_N$，数列满足不递减，即数列中位置靠后的数一定大于或等于位置靠前的数。之后给出M次询问，每次询问包含一个数字Q。对于每次询问，求出数列中第一个大于或等于Q的数字的编号（数字在数列中的编号从1开始）

Input

输入仅有一组数据，共 *M* + 2行。

第一行有两个用空格隔开的自然数 *N* 和 *M*，表示数列中共有*N*个数字，一共有 *M* 次询问。保证 $1 \leq N \leq 10^7, 1 \leq M \leq 1 \times 10^6$ 第二行有 *N* 个用空格隔开的自然数 $a_1, a_2, a_3, \ldots, a_N$，分别表示$a_i$，都有 $1 \leq a_i \leq 10^9, a_i < a_i + 1$

第三行到第*M* + 2行，每行一个数Q，表示要问数列中第一个大于或等于Q的元素的编号。保证 $1 \leq Q \leq a_N$

Output

输出共 *M*行，每行1个自然数，即数列中第一个大于或等于Q的元素的编号

Sample Input

```
5 3
1 3 5 7 9
1
3
6
```

Sample Output

```
1
2
4
```

```
/*
G - 查找[二分]
```

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

const int MAXN = 10000010;
int nums[ MAXN ];
int Query[ MAXN ];

int search( int q, int n ) {
    int len = n;
    int ret = -1;

    int l = 1, r = len;
    while ( len > 0 ) {
        if ( nums[ l ] >= q ) {
            ret = l;
            break;
        }

        int mid = l + ( r - l ) / 2;
        if ( nums[ mid ] < q ) {
            l = mid + 1;
        } else {
            l += 1;
            r = mid;
        }

        len = r - l + 1;
    }
    return ret;
}

int main() {
    int N, M;
    scanf( "%d %d", &N, &M );

    for ( int i = 1; i <= N; ++i )
        scanf( "%d", &nums[ i ] );

    for ( int i = 0; i < M; ++i ) {
        scanf( "%d", &Query[ i ] );
        printf( "%d\n", search( Query[ i ], N ) );
    }

    return 0;
}

/*
input:
5 3
1 3 5 7 9
1
3
6

output:
1
2
```

```
4
*/
```

H : 比k大的数

Time Limit: `2 Sec` Memory Limit: `128 Mb`

Description

给出一个由n个整数组成的数列 $a_1, a_2, a_3, \ldots, a_n$ 之后给出m个整数k，问：对于每一个k，数列中有多少个数比k大?

Input

输入共包括三行

第一行是两个由空格隔开的正整数n和m

第二行是由n个整数组成的数列 $a_1, a_2, a_3, \ldots, a_n$ ，数与数之间由空格隔开。

第三行有m个整数k，数与数之间由空格隔开。

保证 $0 < n \leq 1 \times 10^3; 0 < m \leq 1 \times 10^3;$
$-1 \times 10^3 \leq a_1, a_2, a_3, \ldots, a_n \leq 1 \times 10^3; -1 \times 10^6 \leq k \leq 1 \times 10^6;$

Output

输出共包括m行，每行一个整数，表示数列中有多少个数比k大

Sample Input

```
5 5
1 2 3 4 5
3 5 4 6 -1
```

Sample Output

```
2
0
1
0
5
```

```cpp
/*
H - 比k大的数
*/
#include <bits/stdc++.h>
using namespace std;

// #define DEBUG

int main() {
    int n, m;
    cin >> n >> m;
```

```
    vector<int> nums( n );
    for ( int i = 0; i < n; ++i )
        cin >> nums[ i ];

    sort( nums.begin(), nums.end() );

    vector<int> targets( m );
    for ( int i = 0; i < m; ++i )
        cin >> targets[ i ];

    for ( const auto &t : targets ) {
        auto upper = upper_bound( nums.begin(), nums.end(), t );
        int index =
            ( upper != nums.end() ) ? distance( nums.begin(), upper ) : -1;
        int larger = ( index != -1 ) ? nums.size() - index : 0;

#ifdef DEBUG
        cout << "index=" << index << ", larger=" << larger << endl;
#else
        cout << larger << endl;
#endif
    }

    return 0;
}

/*
input:
5 5
1 2 3 4 5
3 5 4 6 -1

output:
2
0
1
0
5
*/
```

I : 汉诺塔

Time Limit: `2 Sec`, Memory Limit: `1024 Mb`

Description

汉诺塔，是一个源于印度的玩具。大梵天创造世界的时候做了三根金刚石柱子，在一根柱子上从下往上按照大小顺序摞着64片黄金圆盘。大梵天命令婆罗门把圆盘从下面开始按大小顺序重新摆放在另一根柱子上。并且规定，在小圆盘上不能放大圆盘，在三根柱子之间一次只能移动一个圆盘。

现在给你一个n片圆盘的汉诺塔，并从小到大编号为1

至*n*。请你输出搬动*n*个圆盘最少次数的全过程。

- $1 \le n \le 10$

Input

n

Output

输出搬动圆盘最少次数的全过程，格式见案例。

Sample Input

```
3
```

Sample Output

```
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
```

```cpp
/*
I - 汉诺塔
*/
#include <bits/stdc++.h>
using namespace std;

void move( int k, char A, char C ) {
    cout << "Move disk " << k << " from " << A << " to " << C << endl;
}

void hanoi( int k, char A, char B, char C ) {
    if ( k == 1 ) {
        move( k, A, C );
        return;
    }

    hanoi( k - 1, A, C, B );
    move( k, A, C );
    hanoi( k - 1, B, A, C );
}

int main() {
    int n;
    cin >> n;

    hanoi( n, 'A', 'B', 'C' );
    return 0;
}

/*
input:
3
```

```
output:
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
*/
```

J : 切木棍

Time Limit: `2 Sec`, Memory Limit: `1024 Mb`

Description

有*n*根长为，$a_1, a_2 \cdots, a_n$的木棍。

对*n*根木棍总共切*k*次（可以在任意点切割），即最后变成*n* + *k*根木棍。

请输出各种切法种得到的*n* + *k*根木棍中最长那根在各种切法中的最短值（答案需要四舍五入）。

- $1 \leq n \leq 2 \times 10^5$
- $0 \leq k \leq 10^9$
- $1 \leq a_i \leq 10^9$
- 所有值都是整数。

Input

n k

$a_1, a_2 \cdots, a_n$

Output

输出一个整数

Sample Input

```
2 3
7 9
```

Sample Output

```
4
```

Hint

二分

```
/*
J - 切木棍
*/
#include <bits/stdc++.h>
```

```cpp
using namespace std;

int find_min_of_maxlen( vector<int> &rods, int k ) {
    int l = 1, h = rods[ 0 ];
    int ret = -1;

    int len = h - l + 1;
    while ( len > 0 ) {
        if ( l == h ) {
            ret = l;
            break;
        }

        long long mid = l + ( h - l ) / 2;

        long long k_sum = 0;
        for ( int i = 0; i < rods.size(); ++i ) {
            if ( rods[ i ] % mid )
                k_sum += rods[ i ] / mid;
            else
                k_sum += rods[ i ] / mid - 1;
        }

        if ( k_sum <= k ) {
            h = mid;
        } else {
            l = mid + 1;
        }

        len = h - l + 1;
    }

    return ret;
}

int main() {
    int n, k;
    cin >> n >> k;

    vector<int> rods( n );
    for ( int i = 0; i < n; i++ )
        cin >> rods[ i ];

    sort( rods.rbegin(), rods.rend() );

    cout << find_min_of_maxlen( rods, k ) << endl;

    return 0;
}

/*
input:
2 3
7 9

output:
4
*/
```

K : 逆序对

Time Limit: `1 Sec` , Memory Limit: `128 Mb`

Description

对于一个序列a,如果有$a_i > a_j$且$i < j$，则称a_i, a_j为一逆序对。

现给定一个序列，求出序列中逆序对的数量（序列中可能存在重复数字）

Input

第一行是一个整数，表示序列的长度 n。

第二行有 *n* 个整数，第 *i* 个整数表示序列的第 *i* 个数字a_i 。

Output

输出一个整数表示答案。

Sample Input

```
6
5 4 2 6 3 1
```

Sample Output

```
11
```

```cpp
/*
K - 逆序对

reference:
https://leetcode-cn.com/problems/shu-zu-zhong-de-ni-xu-dui-lcof/solution/shu-zu-zhong-de-ni-xu-dui-by-leetcode-solution/
*/

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

int mergeSort( vector<int> &nums, vector<int> &tmp, int l, int r ) {
    if ( l >= r ) {
        return 0;
    }

    int mid = ( l + r ) / 2;
    int inv_count =
        mergeSort( nums, tmp, l, mid ) + mergeSort( nums, tmp, mid + 1, r );
    int i = l, j = mid + 1, pos = l;

    while ( i <= mid && j <= r ) {
        if ( nums[ i ] <= nums[ j ] ) {
            tmp[ pos ] = nums[ i ];
```

```
            ++i;
            inv_count += ( j - ( mid + 1 ) );
        } else {
            tmp[ pos ] = nums[ j ];
            ++j;
        }
        ++pos;
    }

    for ( int k = i; k <= mid; ++k ) {
        tmp[ pos++ ] = nums[ k ];
        inv_count += ( j - ( mid + 1 ) );
    }
    for ( int k = j; k <= r; ++k )
        tmp[ pos++ ] = nums[ k ];

    copy( tmp.begin() + l, tmp.begin() + r + 1, nums.begin() + l );
    return inv_count;
}

int reversePairs( vector<int> &nums ) {
    int n = nums.size();
    vector<int> tmp( n );
    return mergeSort( nums, tmp, 0, n - 1 );
}

int main() {
    int n;
    cin >> n;

    vector<int> nums( n );
    for ( int i = 0; i < n; ++i )
        cin >> nums[ i ];

    cout << reversePairs( nums ) << endl;

    return 0;
}

/*
input:
6
5 4 2 6 3 1

output:
11
*/
```

L：幂运算

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

给你三个整数a,b,q，求 $ab \bmod p$ 的值

Input

第一行是一个整数t，表示t组数据。

接下来的n行，每行有 *3* 个整数，分别表示a, b, p

$t \leq 2 \times 10^5,\ a > 0, b > 0, p \geq 2,$

Output

每组数据，输出一个整数表示答案。

Sample Input

```
2
2 10 9
2 3 3
```

Sample Output

```
7
2
```

```
/*
L - 幂运算

reference:
https://leetcode-cn.com/problems/super-pow/solution/you-qian-ru-shen-kuai-su-mi-suan-fa-xiang-jie-by-l/
*/
#include <bits/stdc++.h>
using namespace std;

long long fastexp( long long a, long long b, long long p ) {
    if ( b == 0 ) return 1;
    a %= p;

    if ( b % 2 == 1 )
        return ( a * fastexp( a, b - 1, p ) ) % p;
    else {
        int res = fastexp( a, b >> 1, p ) % p;
        return ( res * res ) % p;
    }
}

int main() {
    int n;
    cin >> n;

    vector<long long> results( n );
    for ( int i = 0; i < n; ++i ) {
```

```
        long long a, b, p;
        cin >> a >> b >> p;
        long long ans = fastexp( a, b, p );
        results[ i ] = ans;
    }

    for ( const auto &result : results )
        cout << result << endl;

    return 0;
}

/*
input:
2
2 10 9
2 3 3

output:
7
2
*/
```

1017 分治优化

A : A*B

Time Limit: `2 Sec`, Memory Limit: `200 Mb`

Description

如题，计算A*B的值并输出。

Input

两行，分别代表A和B。

A和B的位数不超过2000位。

Output

一行一个整数表示乘积。

Sample Input

```
1
2
```

Sample Output

```
2
```

Hint

普通高精度乘法可以过， 但你也可以试试看分治优化的高精度乘法

- $X = A2^{n/2} + B, Y = C2^{n/2} + D$
- $XY = AC2^n + (AD + BC)2^{n/2} + BD$
- $AD + BC = (A - B)(D - C) + AC + BD$

```
/*
A - AxB

reference:
https://oi-wiki.org/math/poly/fft/
https://paste.ubuntu.com/p/hmK9JR6SgT/
*/
#include <bits/stdc++.h>
using namespace std;

using Complex = std::complex<double>;

const double PI = acos( -1.0 );
const int MAXN = 2e5 + 20;
char a[ MAXN ], b[ MAXN ];
Complex xa[ MAXN ], xb[ MAXN ];

void bit_reverse( Complex N[], int len ) {
    // len should be 2^k
    int i, j, k;
    for ( int i = 1, j = len / 2; i < len - 1; i++ ) {
        if ( i < j ) std::swap( N[ i ], N[ j ] );

        k = len / 2;
        while ( j >= k ) {
            j = j - k;
            k = k / 2;
        }

        if ( j < k ) j += k;
    }
}

void fft( Complex N[], int len, int on ) {
    // on == 1: DFT; on == -1: IDFT; len should be 2^k
    bit_reverse( N, len );
    for ( int h = 2; h <= len; h <<= 1 ) {
        Complex wn( cos( 2 * PI / h ), sin( on * 2 * PI / h ) );
        for ( int j = 0; j < len; j += h ) {
            Complex w( 1, 0 );
            for ( int k = j; k < j + h / 2; k++ ) {
                Complex u = N[ k ];
                Complex t = w * N[ k + h / 2 ];
                N[ k ] = u + t;
```

```
                N[ k + h / 2 ] = u - t;
                w = w * wn;
            }
        }
    }

    if ( on == -1 ) {
        for ( int i = 0; i < len; i++ )
            N[ i ].real( N[ i ].real() / len );
    }
}

int main() {
    while ( scanf( "%s%s", a, b ) != EOF ) {
        int len1 = strlen( a ), len2 = strlen( b );
        int len = 1;
        while ( len < len1 * 2 || len < len2 * 2 )
            len <<= 1;

        for ( int i = 0; i < len1; i++ )
            xa[ i ] = a[ len1 - i - 1 ] - '0';
        for ( int i = 0; i < len2; i++ )
            xb[ i ] = b[ len2 - i - 1 ] - '0';

        for ( int i = len1; i < len; i++ )
            xa[ i ] = 0;
        for ( int i = len2; i < len; i++ )
            xb[ i ] = 0;

        fft( xa, len, 1 );
        fft( xb, len, 1 );

        for ( int i = 0; i < len; i++ )
            xa[ i ] *= xb[ i ];
        fft( xa, len, -1 );

        for ( int i = 0; i < len; i++ ) {
            a[ i ] = int( xa[ i ].real() + 0.5 ) % 10 + '0';
            xa[ i + 1 ].real( xa[ i + 1 ].real() +
                              int( xa[ i ].real() + 0.5 ) / 10 );
        }

        for ( len1 = len - 1; a[ len1 ] == '0' && len1 > 0; len1-- )
            ;
        for ( ; len1 >= 0 && printf( "%c", a[ len1 ] ); len1-- )
            ;
        printf( "\n" );
    }
    return 0;
}

/*
Input:
1
2

Output:
2
```

```
*/
```

B : 矩阵相乘

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

输入两个矩阵，大小均为$n * n$，(n为2的整数次幂）。

矩阵里每个数为自然数且均小于10。

输出两个矩阵相乘的结果。

Input

第一行一个整数：n。 $2 \le n \le 128$

接下来n行，每行n个空格隔开的整数，表示矩阵 $A(i,j)$。

接下来n行，每行n个空格隔开的整数，表示矩阵 $B(i,j)$。

Output

n行，每行n个空格隔开的整数，输出相乘後的矩阵 $C(i,j)$ 的值。

Sample Input

```
2
1 2
3 4
0 1
0 0
```

Sample Output

```
0 1
0 3
```

```
/*
B - 矩阵相乘

refernce:
https://www.cnblogs.com/wuyudong/p/matrix-multiply
 */
#include <bits/stdc++.h>
using namespace std;

using Vec = vector<int>;
using Vec2d = vector<Vec>;

void display( Vec2d &M, int n ) {
    // printf("\n");
    for ( int i = 0; i < n; ++i ) {
        for ( int j = 0; j < n; ++j )
```

```
            printf( "%d ", M[ i ][ j ] );
        printf( "\n" );
    }
}

// 二维方阵乘法
void multiply_2nd( Vec2d &A, Vec2d &B, Vec2d &R ) {
    int M1 = A[ 0 ][ 0 ] * ( B[ 0 ][ 1 ] - B[ 1 ][ 1 ] );
    int M2 = ( A[ 0 ][ 0 ] + A[ 0 ][ 1 ] ) * B[ 1 ][ 1 ];
    int M3 = ( A[ 1 ][ 0 ] + A[ 1 ][ 1 ] ) * B[ 0 ][ 0 ];
    int M4 = A[ 1 ][ 1 ] * ( B[ 1 ][ 0 ] - B[ 0 ][ 0 ] );
    int M5 = ( A[ 0 ][ 0 ] + A[ 1 ][ 1 ] ) * ( B[ 0 ][ 0 ] + B[ 1 ][ 1 ] );
    int M6 = ( A[ 0 ][ 1 ] - A[ 1 ][ 1 ] ) * ( B[ 1 ][ 0 ] + B[ 1 ][ 1 ] );
    int M7 = ( A[ 0 ][ 0 ] - A[ 1 ][ 0 ] ) * ( B[ 0 ][ 0 ] + B[ 0 ][ 1 ] );

    R[ 0 ][ 0 ] = M5 + M4 - M2 + M6;
    R[ 0 ][ 1 ] = M1 + M2;
    R[ 1 ][ 0 ] = M3 + M4;
    R[ 1 ][ 1 ] = M5 + M1 - M3 - M7;
}

void add( int n, Vec2d &A, Vec2d &B, Vec2d &R ) {
    for ( int i = 0; i < n; ++i ) {
        for ( int j = 0; j < n; ++j )
            R[ i ][ j ] = A[ i ][ j ] + B[ i ][ j ];
    }
}

void sub( int n, Vec2d &A, Vec2d &B, Vec2d &R ) {
    for ( int i = 0; i < n; ++i ) {
        for ( int j = 0; j < n; ++j )
            R[ i ][ j ] = A[ i ][ j ] - B[ i ][ j ];
    }
}

void strassen( int n, Vec2d &A, Vec2d &B, Vec2d &R ) {
    int n_ = n / 2;

    Vec2d A11( n_, Vec( n_ ) );
    Vec2d A12( n_, Vec( n_ ) );
    Vec2d A21( n_, Vec( n_ ) );
    Vec2d A22( n_, Vec( n_ ) );

    Vec2d B11( n_, Vec( n_ ) );
    Vec2d B12( n_, Vec( n_ ) );
    Vec2d B21( n_, Vec( n_ ) );
    Vec2d B22( n_, Vec( n_ ) );

    Vec2d R11( n_, Vec( n_ ) );
    Vec2d R12( n_, Vec( n_ ) );
    Vec2d R21( n_, Vec( n_ ) );
    Vec2d R22( n_, Vec( n_ ) );

    Vec2d tempA( n_, Vec( n_ ) );
    Vec2d tempB( n_, Vec( n_ ) );

    Vec2d M1( n_, Vec( n_ ) );
    Vec2d M2( n_, Vec( n_ ) );
```

```
    Vec2d M3( n_, Vec( n_ ) );
    Vec2d M4( n_, Vec( n_ ) );
    Vec2d M5( n_, Vec( n_ ) );
    Vec2d M6( n_, Vec( n_ ) );
    Vec2d M7( n_, Vec( n_ ) );

    if ( n == 2 ) {
        multiply_2nd( A, B, R );
    } else {
        for ( int i = 0; i < n_; ++i ) {
            for ( int j = 0; j < n_; ++j ) {
                A11[ i ][ j ] = A[ i ][ j ];
                A12[ i ][ j ] = A[ i ][ j + n_ ];
                A21[ i ][ j ] = A[ i + n_ ][ j ];
                A22[ i ][ j ] = A[ i + n_ ][ j + n_ ];

                B11[ i ][ j ] = B[ i ][ j ];
                B12[ i ][ j ] = B[ i ][ j + n_ ];
                B21[ i ][ j ] = B[ i + n_ ][ j ];
                B22[ i ][ j ] = B[ i + n_ ][ j + n_ ];
            }
        }

        // M1 = A11(B12 - B22)
        sub( n_, B12, B22, tempB );
        strassen( n_, A11, tempB, M1 );

        // M2 = (A11+A12)B22
        add( n_, A11, A12, tempA );
        strassen( n_, tempA, B22, M2 );

        // M3 = (A21+A22)B11
        add( n_, A21, A22, tempA );
        strassen( n_, tempA, B11, M3 );

        // M4 = A22(B21-B11)
        sub( n_, B21, B11, tempB );
        strassen( n_, A22, tempB, M4 );

        // M5 = (A11+A22)(B11+B22)
        add( n_, A11, A22, tempA );
        add( n_, B11, B22, tempB );
        strassen( n_, tempA, tempB, M5 );

        // M6 = (A12-A22)(B21+B22)
        sub( n_, A12, A22, tempA );
        add( n_, B21, B22, tempB );
        strassen( n_, tempA, tempB, M6 );

        // M7 = (A11-A21)(B11+B12)
        sub( n_, A11, A21, tempA );
        add( n_, B11, B12, tempB );
        strassen( n_, tempA, tempB, M7 );

        // R11 = M5 + M4 - M2 + M6
        add( n_, M5, M4, tempA );
        sub( n_, M6, M2, tempB );
        add( n_, tempA, tempB, R11 );
```

```
        // R12 = M1 + M2
        add( n_, M1, M2, R12 );

        // R21 = M3 + M4
        add( n_, M3, M4, R21 );

        // R22 = M5 + M1 - M3 - M7
        add( n_, M5, M1, tempA );
        add( n_, M3, M7, tempB );
        sub( n_, tempA, tempB, R22 );

        for ( int i = 0; i < n_; ++i ) {
            for ( int j = 0; j < n_; ++j ) {
                R[ i ][ j ] = R11[ i ][ j ];
                R[ i ][ j + n_ ] = R12[ i ][ j ];
                R[ i + n_ ][ j ] = R21[ i ][ j ];
                R[ i + n_ ][ j + n_ ] = R22[ i ][ j ];
            }
        }
    }
}

int main() {
    int n;
    scanf( "%d", &n );

    Vec2d A( n, Vec( n ) );
    for ( int i = 0; i < n; ++i ) {
        for ( int j = 0; j < n; ++j )
            scanf( "%d", &A[ i ][ j ] );
    }

    Vec2d B( n, Vec( n ) );
    for ( int i = 0; i < n; ++i ) {
        for ( int j = 0; j < n; ++j )
            scanf( "%d", &B[ i ][ j ] );
    }

    Vec2d C( n, Vec( n ) );
    strassen( n, A, B, C );
    display( C, n );
    return 0;
}

/*
Sample Input
2
1 2
3 4
0 1
0 0

Sample Output
0 1
0 3
*/
```

C : 数组第 k 大

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

给出 n 个数，以及 k ，求数组中的第 k 大的数。

Input

一个测试文件内有多组数据，每组数据 $n + 1$ 行，第一行正整数 n, k，接下来 n 个正整数 a_i

其中，$1 \leq k \leq n \leq 10^5, 1 \leq a_i \leq 10^9$

Output

对每组数据输出第 k 大的数

Sample Input

```
5 2
1 3 5 7 9
```

Sample Output

```
7
```

```cpp
/*
C - 数组第 k 大
*/
#include <bits/stdc++.h>
using namespace std;

int partition( vector<int> &nums, int low, int high ) {
    int temp = nums[ low ];
    while ( low < high ) {
        while ( low < high && nums[ high ] < temp )
            --high;
        if ( low < high ) {
            nums[ low ] = nums[ high ];
            ++low;
        }

        while ( low < high && nums[ low ] > temp )
            ++low;
        if ( low < high ) {
            nums[ high ] = nums[ low ];
            --high;
        }
    }
    nums[ low ] = temp;
    return low;
}
```

```cpp
int find_maxk( vector<int> &nums, int low, int high, int k ) {
    int pivot = partition( nums, low, high );

    if ( pivot == k - 1 )
        return nums[ k - 1 ];
    else if ( pivot > k - 1 )
        return find_maxk( nums, low, pivot - 1, k );
    else
        return find_maxk( nums, pivot + 1, high, k );
}

int main() {
    int n, k;
    while ( scanf( "%d%d", &n, &k ) != EOF ) {
        vector<int> nums( n );
        for ( int i = 0; i < n; ++i )
            scanf( "%d", &nums[ i ] );

        printf( "%d\n", find_maxk( nums, 0, n - 1, k ) );
    }
    return 0;
}

/*
Sample Input
5 2
1 3 5 7 9

Sample Output
7
*/
```

1021 动态规划入门

A : 打家劫舍问题

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

你是一个专业的小偷，计划偷窃沿街的房屋。每间房内都藏有一定的现金，影响你偷窃的唯一制约因素就是相邻的房屋装有相互连通的防盗系统，如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警。 给定一个代表每个房屋存放金额的非负整数数组，计算你 不触动警报装置的情况下 ，一夜之内能够偷窃到的最高金额。

Input

每组测试案例有两行，第一行只有一个整数N，代表着有N间房屋(2 <= N <= 10000)

第二行有N个整数，代表着每间房屋里的金额，金额范围[0, 1000]。

Output

输出你可以得到的最高金额

Sample Input

```
4
1 3 2 1
5
2 7 9 3 1
```

Sample Output

```
4
12
```

```
/*
A - 打家劫舍问题

reference:
https://leetcode-cn.com/problems/house-robber/solution/da-jia-jie-she-by-
leetcode-solution/
 */
#include <bits/stdc++.h>
using namespace std;

using Array = vector<int>;

int get_max( Array &values ) {
    if ( values.empty() ) return 0;

    if ( 1 == values.size() ) return values[ 0 ];

    int first = values[ 0 ];
    int second = std::max( values[ 0 ], values[ 1 ] );
    for ( int i = 2; i < values.size(); ++i ) {
        int temp = second;
        second = std::max( first + values[ i ], second );
        first = temp;
    }
    return second;
}

int main() {
    int N;
    while ( scanf( "%d", &N ) != EOF ) {
        Array values( N );
        for ( int i = 0; i < N; ++i )
            scanf( "%d", &values[ i ] );

        printf( "%d\n", get_max( values ) );
    }

    return 0;
}
```

```
/*
Sample Input
4
1 3 2 1
5
2 7 9 3 1

Sample Output
4
12
 */
```

B : 最大子序列和

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

给定一个整数数组，找到一个具有最大和的连续子数组（子数组最少包含一个元素），返回其最大和。

Input

有多组测试数据。 对于每组测试数据，第一行只有一个整数N，代表着数组的大小(1 <= N <= 100000)

第二行有N个整数，(-100 <= N <= 100)

Output

每组测试数据仅输出一行，包括一个整数，表示最大子序列和。

Sample Input

```
9
-2 1 -3 4 -1 2 1 -5 4
```

Sample Output

```
6
```

```
/*
B - 最大子序列和
 */
#include <bits/stdc++.h>
using namespace std;

using Array = vector<int>;

int max_seq_sum( Array &nums ) {
    if ( nums.empty() ) return 0;

    if ( 1 == nums.size() ) return nums[ 0 ];

    int seq_max = nums[ 0 ];
```

```
    int max = seq_max;
    for ( int i = 1; i < nums.size(); ++i ) {
        seq_max = std::max( seq_max + nums[ i ], nums[ i ] );
        max = ( seq_max > max ) ? seq_max : max;
    }
    return max;
}

int main() {
    int N;
    while ( scanf( "%d", &N ) != EOF ) {
        Array nums( N );
        for ( int i = 0; i < N; ++i )
            scanf( "%d", &nums[ i ] );

        printf( "%d\n", max_seq_sum( nums ) );
    }
    return 0;
}

/*
Sample Input
9
-2 1 -3 4 -1 2 1 -5 4

Sample Output
6
 */
```

C：斐波那契数列

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

斐波那契数列指的是这样一个数列：0、1、1、2、3、5、8、13、21、34、……在数学上，斐波那契数列以如下被以递推的方法定义：$F(0)=0$，$F(1)=1$, $F(n)=F(n-1)+F(n-2)$（$n \geq 2$，$n \in N^*$）

Input

多组数据。对于每组数据：输入两个正整数a, b(0 <= a <= b <= 20).

Output

对于每组输入,输出一行,包含一个整数，求F(b) -F(a)的值。

Sample Input

```
0 1
2 3
3 4
```

Sample Output

```
1
1
1
```

```
/*
C - 斐波那契数列

reference:
https://leetcode-cn.com/problems/fei-bo-na-qi-shu-lie-lcof/solution/fei-bo-na-qi-shu-lie-by-leetcode-solutio-hbss/
 */
#include <bits/stdc++.h>
using namespace std;

int fib( int n ) {
    if ( n < 2 ) return n;

    int p = 0, q = 0, r = 1;
    for ( int i = 2; i <= n; ++i ) {
        p = q;
        q = r;
        r = p + q;
    }
    return r;
}

int main() {
    int a, b;
    while ( scanf( "%d%d", &a, &b ) != EOF ) {
        printf( "%d\n", fib( b ) - fib( a ) );
    }
    return 0;
}

/*
Sample Input
0 1
2 3
3 4

Sample Output
1
1
1
*/
```

D：矩阵链相乘

Time Limit: `5 Sec`, Memory Limit: `128 Mb`

Description

设A1,A2,...,An为矩阵序列，Ai是阶为Pi−1∗Pi的矩阵(1≤i≤n)。

试确定矩阵的乘法顺序，使得计算A1 A2 ... An 过程中元素相乘的总次数最少。

Input

多组数据

第一行一个整数n(n <= 300)，表示一共有n个矩阵。

第二行n个整数B1, B2, B3... Bn(Bi <= 100)，第i个数Bi表示第i个矩阵的行数和第i-1个矩阵的列数。

等价地，可以认为第j个矩阵Aj(1 <= j <= n)的行数为Bj，列数为Bj+1。

Output

对于每组数据，输出一个最优计算次数

Sample Input

```
5
74 16 58 58 88 80
5
10 1 50 50 20 5
```

Sample Output

```
342848
3650
```

```cpp
/*
D - 矩阵链相乘
 */
#include <bits/stdc++.h>
using namespace std;

// #define DEBUG

using Array = vector<int>;
using Matrix = vector<Array>;

const int MAXN = 310;

Matrix m( MAXN, Array( MAXN ) );
Matrix s( MAXN, Array( MAXN ) );

void matrix_chain( Array &arr, int n ) {
    if ( arr.empty() ) return;

    for ( int r = 2; r <= n; ++r ) {
        for ( int i = 1; i <= ( n - r + 1 ); ++i ) {
            int j = i + r - 1;
            m[ i ][ j ] = m[ i + 1 ][ j ] + arr[ i - 1 ] * arr[ i ] * arr[ j ];
            s[ i ][ j ] = i;
```

```
            for ( int k = i + 1; k < j; ++k ) {
                int temp = m[ i ][ k ] + m[ k + 1 ][ j ] +
                           arr[ i - 1 ] * arr[ k ] * arr[ j ];
                if ( temp < m[ i ][ j ] ) {
                    m[ i ][ j ] = temp;
                    s[ i ][ j ] = k;
                }
            }
        }
    }

#ifdef DEBUG
    for ( int i = 1; i <= n; ++i ) {
        for ( int j = 1; j <= n; ++j )
            printf( "%8d", m[ i ][ j ] );
        printf( "\n" );
    }
#endif
}

int main() {
    int n;
    while ( scanf( "%d", &n ) != EOF ) {
        Array arr( n + 1 );
        for ( int i = 0; i <= n; ++i )
            scanf( "%d", &arr[ i ] );

        matrix_chain( arr, n );
        printf( "%d\n", m[ 1 ][ n ] );
    }
    return 0;
}

/*
Sample Input
5
74 16 58 58 88 80
5
10 1 50 50 20 5

Sample Output
342848
3650
 */
```

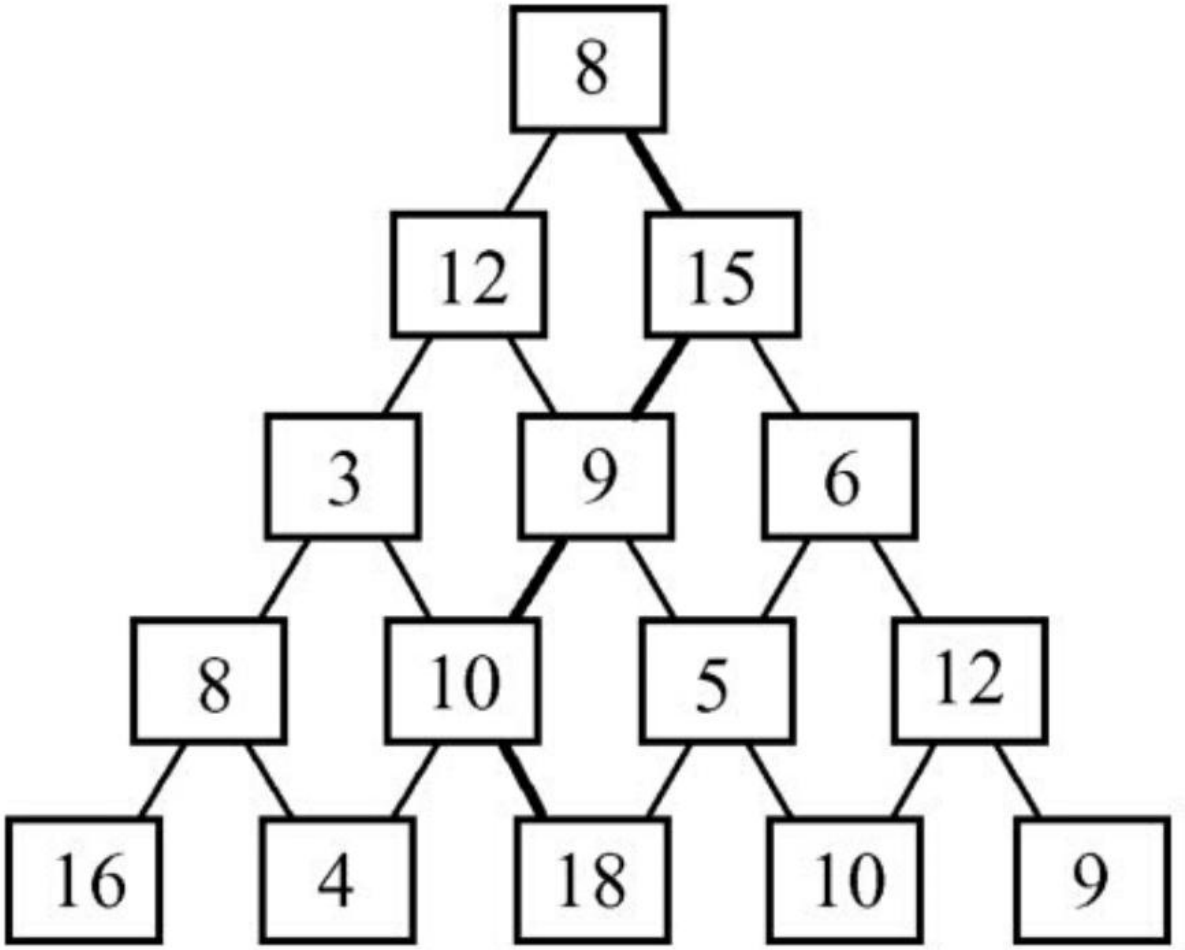
1025 经典动态规划

A : 数字三角形

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

给出如下图的数字三角形，要求从顶层走到底层，若每一步只能走到相邻的结点，则经过的结点的数字之和最大是多少？



Input

输入数据首先包括一个整数T,表示测试实例的个数。

每个测试实例的第一行是一个整数N(1 <= N <= 100)，表示数塔的高度。

接下来用N行数字表示数塔，其中第i行有个i个整数，且所有的整数均在区间[0,99]内。

Output

对于每个测试实例，输出可能得到的最大和，每个实例的输出占一行。

Sample Input

```
1
5
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
```

Sample Output

```
30
```

```cpp
/*
A - 数字三角形
 */
#include <bits/stdc++.h>
using namespace std;

// #define DEBUG

using Array = vector<int>;

void display( Array &nums, int N ) {
    for ( int i = N; i >= 1; --i ) {
        for ( int j = ( i - 1 ) * i / 2; j < ( i + 1 ) * i / 2; ++j )
            printf( "%d ", nums[ j ] );
        printf( "\n" );
    }
}

int get_max_sum( Array &nums, int N ) {
    if ( nums.empty() ) return 0;
    if ( nums.size() == 1 ) return nums[ 0 ];

    for ( int i = N - 1; i >= 1; --i ) {
        for ( int j = ( i - 1 ) * i / 2; j < ( i + 1 ) * i / 2; ++j ) {
            int left = j + i;
            int right = left + 1;
            nums[ j ] += std::max( nums[ left ], nums[ right ] );
        }
    }

#ifdef DEBUG
    display( nums, N );
#endif

    return nums[ 0 ];
}

int main() {
    int T, N;
    scanf( "%d", &T );
    while ( T-- > 0 && scanf( "%d", &N ) != EOF ) {
        int len = ( N + 1 ) * N / 2;
        Array nums( len );
        for ( int i = 0; i < len; ++i )
            scanf( "%d", &nums[ i ] );

        printf( "%d\n", get_max_sum( nums, N ) );
    }

    return 0;
}
```

```
/*
Sample Input
1
5
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5

Sample Output
30
 */
```

B : 最长递增子序列 I

Time Limit: `3 Sec`, Memory Limit: `128 Mb`

Description

给出长度为N的数组，找出这个数组的最长递增子序列。(递增子序列是指，子序列的元素是递增的)

例如: 1 3 2 5 4 7 6 9 8, 最长递增子序列为1 3 5 7 9

Input

输入数据首先包括一个整数T(1 <= 10),表示测试实例的个数。

每个测试实例的第一行是一个整数N(2 <= N <= 5000)，表示序列的长度。

第二行数字是一组数组，且所有的整数均在区间[0,1,000,000]内。

Output

对于每个测试实例，输出最长递增子序列的长度，每个实例的输出占一行。

Sample Input

```
1
9
1 3 2 5 4 7 6 9 8
```

Sample Output

```
5
```

```
/*
B - 最长递增子序列 I

reference:
https://leetcode-cn.com/problems/longest-increasing-subsequence/solution/zui-chang-shang-sheng-zi-xu-lie-by-leetcode-soluti/
 */
#include <bits/stdc++.h>
```

```cpp
using namespace std;

using Array = vector<int>;

int max_sub_len( Array &nums ) {
    if ( nums.empty() ) return 0;

    int n = nums.size();
    Array dp( n );
    for ( int i = 0; i < n; ++i ) {
        dp[ i ] = 1;
        for ( int j = 0; j < i; ++j ) {
            if ( nums[ j ] < nums[ i ] )
                dp[ i ] = std::max( dp[ i ], dp[ j ] + 1 );
        }
    }
    return *std::max_element( dp.begin(), dp.end() );
}

int main() {
    int T, N;
    scanf( "%d", &T );
    while ( T-- && scanf( "%d", &N ) != EOF ) {
        Array nums( N );
        for ( int i = 0; i < N; ++i )
            scanf( "%d", &nums[ i ] );

        printf( "%d\n", max_sub_len( nums ) );
    }

    return 0;
}

/*
Sample Input
1
9
1 3 2 5 4 7 6 9 8

Sample Output
5
 */
```

C：最长公共子序列

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

求两个序列的最每组测试样例都为一行，两组字符串，每组不超过1000，用空格隔开。求最长公共子序列，都为小写字母。

Input

每组测试样例都为一行，两组字符串，每组不超过1000，用空格隔开。

Output

对于每个测试实例，输出最长公共子序列的长度，每个实例的输出占一行。

Sample Input

```
abcfbc abfcab
programming contest
abcd mnp
```

Sample Output

```
4
2
0
```

```cpp
/*
C - 最长公共子序列
 */
#include <bits/stdc++.h>
using namespace std;

using Array = vector<int>;
using Matrix = vector<Array>;

int LCS( string &str1, string &str2 ) {
    if ( str1.empty() || str2.empty() ) return 0;

    int n1 = str1.size(), n2 = str2.size();
    Matrix C( n1 + 1, Array( n2 + 1 ) );

    for ( int i = 1; i <= n1; ++i ) {
        for ( int j = 1; j <= n2; ++j ) {
            if ( str1[ i - 1 ] == str2[ j - 1 ] )
                C[ i ][ j ] = C[ i - 1 ][ j - 1 ] + 1;
            else if ( C[ i - 1 ][ j ] >= C[ i ][ j - 1 ] )
                C[ i ][ j ] = C[ i - 1 ][ j ];
            else
                C[ i ][ j ] = C[ i ][ j - 1 ];
        }
    }

    return C[ n1 ][ n2 ];
}

int main() {
    string str1, str2;
    while ( cin >> str1 >> str2 ) {
        cout << LCS( str1, str2 ) << endl;
    }
    return 0;
```

```
}

/*
Sample Input
abcfbc abfcab
programming contest
abcd mnp

Sample Output
4
2
0
 */
```

D：连续数组最大和问题

Time Limit: `1 Sec`， Memory Limit: `128 Mb`

Description

求最大连续子段和，并输出此子段的起始位置和终止位置的值。

例如给定序列{ -2, 11, -4, 13, -5, -2 }，其最大连续子序列为{ 11, -4, 13 }，最大和为20。

Input

测试输入包含若干测试用例，每个测试用例占2行，第1行给出正整数K(< 10000)，第2行给出K个整数，中间用空格分隔。当K为0时，输入结束，该用例不被处理。

Output

对每个测试用例，在1行里输出最大和、最大连续子序列的第一个和最后一个元 素，中间用空格分隔。如果最大连续子序列不唯一，则输出序号i和j最小的那个（如输入样例的第2、3组）。若所有K个元素都是负数，则定义其最大和为0，输出整个序列的首尾元素。

Sample Input

```
6
-2 11 -4 13 -5 -2
10
-10 1 2 3 4 -5 -23 3 7 -21
6
5 -8 3 2 5 0
1
10
3
-1 -5 -2
3
-1 0 -2
0
```

Sample Output

```
20 11 13
10 1 4
10 3 5
10 10 10
0 -1 -2
0 0 0
```

```
/*
D - 连续数组最大和问题
 */
#include <bits/stdc++.h>
using namespace std;

using Array = vector<int>;

void find_max_seq( Array &nums ) {

    int n = nums.size();
    Array S( n ), E( n ), dp( n );

    dp[ 0 ] = nums[ 0 ];
    for ( int i = 1; i < n; ++i ) {
        if ( dp[ i - 1 ] <= 0 ) {
            S[ i ] = i;
            E[ i ] = i;
            dp[ i ] = nums[ i ];
        } else {
            S[ i ] = S[ i - 1 ];
            E[ i ] = i;
            dp[ i ] = dp[ i - 1 ] + nums[ i ];
        }
    }

    int best_index = 0;
    int best = dp[ 0 ];
    for ( int i = 1; i < n; ++i ) {
        if ( dp[ i ] > best ) {
            best = dp[ i ];
            best_index = i;
        }
    }

    int start = nums[ S[ best_index ] ];
    int end = nums[ E[ best_index ] ];
    printf( "%d %d %d\n", dp[ best_index ], start, end );
}

int main() {
    int N;
    while ( scanf( "%d", &N ) != EOF && N != 0 ) {
        bool isAllNegative = true;

        Array nums( N );
        for ( int i = 0; i < N; ++i ) {
```

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

            if ( nums[ i ] >= 0 ) isAllNegative = false;
        }

        if ( isAllNegative )
            printf( "%d %d %d\n", 0, nums[ 0 ], nums[ N - 1 ] );
        else
            find_max_seq( nums );
    }
    return 0;
}

/*
Sample Input
6
-2 11 -4 13 -5 -2
10
-10 1 2 3 4 -5 -23 3 7 -21
6
5 -8 3 2 5 0
1
10
3
-1 -5 -2
3
-1 0 -2
0

Sample Output
20 11 13
10 1 4
10 3 5
10 10 10
0 -1 -2
0 0 0
 */
```

E : 01饭卡

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

如果购买一个商品之前，卡上的剩余金额大于或等于5元，就一定可以购买成功（即使购买后卡上余额为负），否则无法购买（即使金额足够）。所以大家都希望尽量使卡上的余额最少。 某天，食堂中有n种菜出售，每种菜可购买一次。已知每种菜的价格以及卡上的余额，问最少可使卡上的余额为多少。

Input

多组数据。对于每组数据： 第一行为正整数n，表示菜的数量。n<=1000。 第二行包括n个正整数，表示每种菜的价格。价格不超过50。 第三行包括一个正整数m，表示卡上的余额。m<=1000。

n=0表示数据结束。

Output

对于每组输入,输出一行,包含一个整数，表示卡上可能的最小余额。

Sample Input

```
1
50
5
10
1 2 3 2 1 1 2 3 2 1
50
0
```

Sample Output

```
-45
32
```

```
/*
E - 01饭卡

reference:
https://blog.51cto.com/u_12312066/3632308
https://docs.qq.com/pdf/DRHdIcGltTllSVVVp
 */
#include <bits/stdc++.h>
using namespace std;

using Array = vector<int>;

int ZeroOnePack( Array &prices, int total ) {
    int n = prices.size();
    Array dp( total + 1 );

    std::sort( prices.begin(), prices.end() );

    for ( int i = 1; i < n; ++i ) {
        for ( int j = total - 5; j >= prices[ i - 1 ]; --j )
            dp[ j ] = std::max( dp[ j ],
                                dp[ j - prices[ i - 1 ] ] + prices[ i - 1 ] );
    }

    return total - dp[ total - 5 ] - prices[ n - 1 ];
}

int main() {
    int n, m;
    while ( scanf( "%d", &n ) != EOF && n != 0 ) {
        Array prices( n );
        for ( int i = 0; i < n; ++i )
            scanf( "%d", &prices[ i ] );

        scanf( "%d", &m );
```

```
        if ( m < 5 )
            printf( "%d\n", m );
        else
            printf( "%d\n", ZeroOnePack( prices, m ) );
    }

    return 0;
}

/*
Sample Input
1
50
5
10
1 2 3 2 1 1 2 3 2 1
50
0

Sample Output
-45
32
 */
```

F : 合并石子

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

在一个圆形操场的四周摆放 N 堆石子,现要将石子有次序地合并成一堆.规定每次只能选相邻的2堆合并成新的一堆，并将新的一堆的石子数，记为该次合并的得分。

试设计出一个算法,计算出将 N堆石子合并成 1堆的最小得分和最大得分。

Input

数据的第 1行是正整数 N，表示有 N 堆石子, N <= 500。

第 2 行有 N 个整数，第 i个整数 ai 表示第 i 堆石子的个数, ai <= 50。

Output

输出共 2 行，第 1 行为最小得分，第 2 行为最大得分。

Sample Input

```
4
4 5 9 4
```

Sample Output

```
43
54
```

```
/*
F - 合并石子

reference:
https://www.cnblogs.com/onetrainee/p/11955567.html
https://blog.csdn.net/weixin_43899069/article/details/111469996
 */
#include <bits/stdc++.h>
using namespace std;

using Array = vector<int>;
using Matrix = vector<Array>;

const int INF = 0x3f3f3f3f;
const int MAXN = 1010;

Matrix dp_min( MAXN, Array( MAXN, INF ) );
Matrix dp_max( MAXN, Array( MAXN ) );
Array costs( MAXN );

void merge( Array &arr, int n ) {
    for ( int i = 1; i <= 2 * n; ++i )
        costs[ i ] = costs[ i - 1 ] + arr[ i ];

    for ( int len = 2; len <= n; ++len ) {
        for ( int i = 1; len + i - 1 <= 2 * n; ++i ) {
            int j = len + i - 1;
            for ( int k = i; k < j; ++k ) {
                dp_min[ i ][ j ] = std::min(
                    dp_min[ i ][ j ], dp_min[ i ][ k ] + dp_min[ k + 1 ][ j ] +
                                          costs[ j ] - costs[ i - 1 ] );
                dp_max[ i ][ j ] = std::max(
                    dp_max[ i ][ j ], dp_max[ i ][ k ] + dp_max[ k + 1 ][ j ] +
                                          costs[ j ] - costs[ i - 1 ] );
            }
        }
    }

    int ans_min = INF;
    int ans_max = -1;
    for ( int i = 1; i <= n; i++ ) {
        ans_min = min( ans_min, dp_min[ i ][ i + n - 1 ] );
        ans_max = max( ans_max, dp_max[ i ][ i + n - 1 ] );
    }

    printf( "%d\n", ans_min );
    printf( "%d\n", ans_max );
}

int main() {
    int N;
```

```
    scanf( "%d", &N );

    Array arr( 2 * N + 1 );
    for ( int i = 1; i <= N; ++i ) {
        scanf( "%d", &arr[ i ] );
        arr[ i + N ] = arr[ i ];
        dp_min[ i ][ i ] = 0;
        dp_min[ i + N ][ i + N ] = 0;
    }

    merge( arr, N );

    return 0;
}

/*
Sample Input
4
4 5 9 4

Sample Output
43
54
 */
```

1028 贪心入门

A : 买最多

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

给出物品单价和重量，给出当前金额，问最多能买多重的物品，物品可拆。

Input

输入数据第一行包含一个正整数t($t \le 100$)，代表共有t组测试样例。每组测试样例的第一行包含两个正整数n和m($1 \le n, m \le 1500$)，分别代表有n块钱，有m种物品。接下来输入m行，每行包含两个正整数a，b($1 \le a \le 25, 1 \le b \le 10$)，分别代表当前物品的单价和重量。可认为当前金额买不了全部物品

Output

对于每组输出占一行，保留2位小数。

Sample Input

```
1
6 3
4 2
3 2
2 2
```

Sample Output

```
2.67
```

```
/*
A - 买最多
 */
#include <bits/stdc++.h>
using namespace std;

// #define DEBUG

using Pair = pair<int, int>;
using Item = vector<Pair>;

struct {
    bool operator()( const Pair &a, const Pair &b ) const {
        return a.first < b.first;
    }
} compare_first;

int main() {
    int t;
    scanf( "%d", &t );
    while ( t-- ) {
        int n, m; // n-total money, m-kinds of items
        scanf( "%d%d", &n, &m );

        Item item( m );
        for ( int i = 0; i < m; ++i )
            scanf( "%d%d", &item[ i ].first,
                   &item[ i ].second ); // first-price, second-total weight

        std::sort( item.begin(), item.end(),
                   compare_first ); // sort by item price

#ifdef DEBUG
        printf( "\n" );
        for ( int i = 0; i < m; ++i )
            printf( "%d %d\n", item[ i ].first, item[ i ].second );
        printf( "\n" );
#endif

        double total_weight = 0.0;
        double money_left = n;
        for ( int i = 0; money_left > 0 && i < m; ++i ) {
            double w = money_left /
                       item[ i ].first; // the money left can buy w item[i]
            w = ( w < item[ i ].second ) ? w : item[ i ].second;
            total_weight += w;
            money_left -= w * item[ i ].first;

#ifdef DEBUG
            printf( "[%d] %lf %lf | %lf\n", i, w, money_left, total_weight );
#endif
        }
```

```
        printf( "%.2lf\n", total_weight );
    }
    return 0;
}

/*
Sample Input
1
6 3
4 2
3 2
2 2

Sample Output
2.67
 */
```

B : XHD来寻宝

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

XHD去寻宝，找到多种宝贝，每种宝贝单位体积的价格不一样，现在请你帮忙计算XHD最多能带多少价值的宝贝？（假设宝贝可以分割，分割后的价值和对应的体积成正比）

Input

输入包含多个测试实例，每个实例的第一行是两个整数v和n(v,n<1000)，分别表示口袋的容量和宝贝的种类，接着的n行每行包含2个整数pi和mi(0<pi,mi<1000)，分别表示某种宝贝的单价和对应的体积，v为0的时候结束输入。

Output

对于每个测试实例，请输出XHD最多能取回多少价值的宝贝，每个实例的输出占一行。

Sample Input

```
2 2
3 1
2 3
0
```

Sample Output

```
5
```

```
/*
B - XHD来寻宝
 */
#include <bits/stdc++.h>
using namespace std;
```

```cpp
// #define DEBUG

using Pair = pair<int, int>;
using Item = vector<Pair>;

struct {
    bool operator()( const Pair &a, const Pair &b ) const {
        return a.first < b.first;
    }
} compare_first;

int main() {
    int v; // capacity of pocket
    while ( scanf( "%d", &v ) != EOF && v != 0 ) {
        int n; // kinds of treasure
        scanf( "%d", &n );

        Item item( n );
        for ( int i = 0; i < n; ++i )
            scanf( "%d%d", &item[ i ].first,
                   &item[ i ].second ); // first-price, second-volume

        std::sort( item.rbegin(), item.rend(),
                   compare_first ); // sort by item price

#ifdef DEBUG
        printf( "\n" );
        for ( int i = 0; i < n; ++i )
            printf( "%d %d\n", item[ i ].first, item[ i ].second );
        printf( "\n" );
#endif

        int total_value = 0;
        for ( int i = 0; v > 0 && i < n; ++i ) {
            int w = ( v < item[ i ].second )
                        ? v
                        : item[ i ].second; // the volume can take
            total_value += item[ i ].first * w;
            v -= w;

#ifdef DEBUG
            printf( "[%d] %d %d | %d\n", i, w, v, total_value );
#endif
        }
        printf( "%d\n", total_value );
    }
    return 0;
}

/*
Sample Input
2 2
3 1
2 3
0

Sample Output
5
```

```
 */
```

C : 最少拦截系统

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

某国为了防御敌国的导弹袭击,发展出一种导弹拦截系统.但是这种导弹拦截系统有一个缺陷:虽然它的第一发炮弹能够到达任意的高度,但是以后每一发炮弹都不能超过前一发的高度.某天,雷达捕捉到敌国的导弹来袭.

由于成本所在，我们要用尽可能少的拦截系统，请帮忙算一下最少需要多少套拦截系统方可抵御一次进攻。

Input

输入若干组数据.每组数据包括:导弹总个数,导弹依次飞来的高度。

第一个数输入n(1≤n≤100)，代表有n个导弹，接下来在同一行中，输入n个数，代表导弹依次飞来的高度。

Output

对应每组数据输出拦截所有导弹最少要配备多少套这种导弹拦截系统.

Sample Input

```
8 389 207 155 300 299 170 158 65
```

Sample Output

```
2
```

Hint

a不能超过b的意思是a≤b

```cpp
/*
C - 最少拦截系统

reference:
https://blog.csdn.net/hurmishine/article/details/52926957
 */
#include <bits/stdc++.h>
using namespace std;

using Array = vector<int>;

int main() {
    int n;
    while ( scanf( "%d", &n ) != EOF ) {
        Array items( n );
```

```
        int x;
        int cnt = 1;
        bool flag = false;
        scanf( "%d", &items[ 0 ] );

        for ( int i = 1; i < n; ++i ) {
            flag = false;
            scanf( "%d", &x );
            for ( int j = 0; j <= cnt; ++j ) {
                if ( items[ j ] >= x ) {
                    items[ j ] = x;
                    flag = true;
                    break;
                }
            }
            if ( !flag ) items[ ++cnt ] = x;
        }

        printf( "%d\n", cnt );
    }

    return 0;
}

/*
Sample Input
8 389 207 155 300 299 170 158 65

Sample Output
2
 */
```

D : 单设备区间调度

Time Limit: `1 Sec` Memory Limit: `128 Mb`

Description

给出调度任务的开始时间和结束时间*l*,*r*，同一时间段仅能调度一个任务，要求最多能调度多少个任务

Input

多组输入，每组输入第一行包含一个正整数*t*($t \le 100$)，代表共有*t*个调度任务。接下来输入*t*行，每一行包含两个正整数*l*和*r*($0 \le l < r \le 10^9$)，分别一个调度任务的开始时间和结束时间*l*,*r*。当*t* = 0时，代表输入结束，不做处理。

Output

对于每个测试实例，输出能调度的最多任务的个数，每个实例的输出占一行

Sample Input

```
5
1 3
3 6
2 4
4 7
6 9
0
```

Sample Output

```
3
```

```
/*
D - 单设备区间调度
 */
#include <bits/stdc++.h>
using namespace std;

// #define DEBUG

using Pair = pair<int, int>;
using Item = vector<Pair>;
using Array = vector<int>;

struct {
    bool operator()( const Pair &a, const Pair &b ) const {
        return a.second < b.second;
    }
} compare_second;

int main() {
    int t;
    while ( scanf( "%d", &t ) != EOF && t != 0 ) {
        Item items( t );
        for ( int i = 0; i < t; ++i )
            scanf( "%d%d", &items[ i ].first, &items[ i ].second );

        std::sort( items.begin(), items.end(), compare_second );

#ifdef DEBUG
        printf( "\n" );
        for ( int i = 0; i < t; ++i )
            printf( "%d %d\n", items[ i ].first, items[ i ].second );
        printf( "\n" );
#endif

        int cnt = 0;
        Array dp( t, -1 );
        for ( int i = 0; i < t; ++i ) {
            if ( items[ i ].first >= dp[ cnt ] ) {
                ++cnt;
                dp[ cnt ] = items[ i ].second;
#ifdef DEBUG
```

```
                printf( "[%d] %d (%d %d)\n", cnt, dp[ cnt ], items[ i ].first,
                        items[ i ].second );
#endif
            }
        }
        printf( "%d\n", cnt );
    }
    return 0;
}

/*
Sample Input
5
1 3
3 6
2 4
4 7
6 9
0

Sample Output
3
 */
```

1032 经典贪心

A : 多设备区间调度

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

多个调度任务，给出调度任务的起始时间和结束时间，问至少需要多少个设备能完成调度任务，在保证最小设备的情况，要求所有设备运行总时长最短。

每个设备的运行时长：从给这个设备的第一个任务开始计算到这个设备的最后一个任务结束，中间哪怕没有任务也要计算运行时长。

Input

输入的第一行包含一个正整数t($t \le 100$)，代表共有t组输入实例。每组输入实例中，第一行包含一个正整数n($n \le 100$)，接下来输入n行，每一行包含两个正整数l和r($0 \le l < r \le 109$)，分别一个调度任务的开始时间和结束时间l,r。读入至文件结束为止。

Output

每组实例输出两个正整数，分别代表使用设备的数量，设备运行的总时长，每组实例输出占一行

Sample Input

```
1
5
1 3
3 6
2 4
5 7
6 9
```

Sample Output

```
2 13
```

```cpp
/*
A - 多设备区间调度
 */
#include <bits/stdc++.h>
using namespace std;

using Pair = pair<int, int>;
using Item = vector<Pair>;

struct {
    bool operator()( const Pair &a, const Pair &b ) const {
        return a.first < b.first;
    }
} compare_first;

void schedule( const Item &items ) {
    int n = items.size();

    multiset<int> s;
    long long total = 0;
    for ( int i = 0; i < n; ++i ) {
        auto it = s.begin();
        if ( !s.empty() && items[ i ].first >= *it ) {
            it = --s.upper_bound( items[ i ].first );
            total += items[ i ].second - *it;
            s.erase( it );
            s.insert( items[ i ].second );
        } else {
            total += items[ i ].second - items[ i ].first;
            s.insert( items[ i ].second );
        }
    }
    printf( "%lu %lld\n", s.size(), total );
}

int main() {
    int t;
    scanf( "%d", &t );
    while ( t-- ) {
        int n;
        scanf( "%d", &n );

        Item devices( n );
```

```
        for ( int i = 0; i < n; i++ )
            scanf( "%d%d", &devices[ i ].first, &devices[ i ].second );

        sort( devices.begin(), devices.end(), compare_first );

        schedule( devices );
    }
    return 0;
}

/*
Sample Input
1
5
1 3
3 6
2 4
5 7
6 9

Sample Output
2 13
 */
```

B : 找零钱

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

现有100，50，20，10，5，2，1元的纸币，现在一个物品价值n元，问至少需要多少张钱，才可以支付该物品。

Input

输入多组数据，每组第一行输入整数n（1≤n≤9999）。

Output

输出至少需要的钱的张数，并输出方案（输出格式：币值`*`张数+币值`*`张数+....=n,币值从大到小输出，其中张数为1则无需乘以张数）

Sample Input

```
6
1
1000
```

Sample Output

```
2 5+1=6
1 1=1
10 100*10=1000
```

```cpp
/*
B - 找零钱
 */
#include <bits/stdc++.h>
using namespace std;

const vector<int> moneys = { 100, 50, 20, 10, 5, 2, 1 };
const int SIZE = moneys.size();

int main() {
    int n;
    while ( scanf( "%d", &n ) != EOF ) {
        vector<int> nums( SIZE );

        int remain = n;
        int total_num = 0;
        for ( int i = 0; i < SIZE; ++i ) {
            nums[ i ] = remain / moneys[ i ];
            remain %= moneys[ i ];
            total_num += nums[ i ];
        }

        printf( "%d ", total_num );

        bool started = false;
        for ( int i = 0; i < SIZE; ++i ) {
            if ( nums[ i ] > 0 ) {
                if ( !started )
                    started = true;
                else
                    printf( "+" );

                printf( "%d", moneys[ i ] );

                if ( nums[ i ] > 1 ) printf( "*%d", nums[ i ] );
            }
        }
        printf( "=%d\n", n );
    }

    return 0;
}

/*
Sample Input
6
1
1000

Sample Output
2 5+1=6
1 1=1
10 100*10=1000
 */
```

C：最小延迟调度问题

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

假定有一单个的资源在一个时刻只能处理一个任务。现给定一组任务，其中的每个任务 i 包含一个持续时间 t_i 和截止时间 d_i 。设计与实现一个算法，从 0 时刻开始任务，对这组任务给出一个最优调度方案，使其所有任务中的最大延迟最小化。任务 i的延迟指实际完成时间 f_i 减去截止时间 d_i

Input

第一行输入一个t，代表有t组样例。（t≤10）

每个样例第一行输入一个n（n≤100），代表工作数，接下来n行，每行输入两个数ti（1≤ti≤100）和di（1≤di≤1000），代表任务持续时间和截止时间。

Output

输出所有任务中的最大延迟的最小值

Sample Input

```
1
6
3 6
2 8
1 9
4 9
3 14
2 15
```

Sample Output

```
1
```

```cpp
/*
C - 最小延迟调度问题
 */
#include <bits/stdc++.h>
using namespace std;

// #define DEBUG

using Pair = pair<int, int>;
using Item = vector<Pair>;

struct {
    bool operator()( const Pair &a, const Pair &b ) const {
        return a.second < b.second;
    }
} compare_second;

int main() {
    int t;
```

```
    scanf( "%d", &t );
    while ( t-- ) {
        int n;
        scanf( "%d", &n );

        Item jobs( n );
        for ( int i = 0; i < n; ++i )
            scanf( "%d%d", &jobs[ i ].first, &jobs[ i ].second );

        std::sort( jobs.begin(), jobs.end(), compare_second );

        int finish_time = 0;
        int max_time = 0;
        for ( int i = 0; i < n; ++i ) {
            finish_time += jobs[ i ].first;
            int latency = finish_time - jobs[ i ].second;

            if ( latency > 0 )
                max_time = ( latency > max_time ) ? latency : max_time;
        }

        printf( "%d\n", max_time );
    }

    return 0;
}

/*
Sample Input
1
6
3 6
2 8
1 9
4 9
3 14
2 15

Sample Output
1
 */
```

D : 最优装载

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

有一个载重为n的大卡车，有m个货物要运送，给出货物的重量。假设卡车无限大，载货量仅受载重量限制，问最多一次能载多少个货物

Input

输入第一行包含一个正整数t($t \le 104$)，代表共有t组实例。

每组实例第一行包含两个正整数n和m($1 \le n, m \le 104$)，分别代表卡车的载重和货物数量

接下来m行，每行包含一个正整数x($1 \le x \le 50$)，代表第i个物品的重量

Output

对于每组实例，每行仅输出一个正整数，表示最多一次能装载的货物数量。

Sample Input

```
1
10 4
3
4
5
6
```

Sample Output

```
2
```

```cpp
/*
D - 最优装载
 */
#include <bits/stdc++.h>
using namespace std;

int main() {
    int t;
    scanf( "%d", &t );
    while ( t-- ) {
        int n, m;
        scanf( "%d%d", &n, &m );

        vector<int> w( m );
        for ( int i = 0; i < m; i++ )
            scanf( "%d", &w[ i ] );

        std::sort( w.begin(), w.end() );

        int ans = 0;
        for ( int i = 0; i < m; ++i ) {
            if ( n >= w[ i ] ) {
                n -= w[ i ];
                ++ans;
            }
        }
        printf( "%d\n", ans );
    }
    return 0;
}
```

```
/*
Sample Input
1
10 4
3
4
5
6

Sample Output
2
 */
```

1036 回溯入门

A : 马的遍历

Time Limit: `1 Sec` Memory Limit: `128 Mb`

Description

给出标准 8×8 国际象棋棋盘上的两个格子，马的移动方式为 “日” 字形（如下图所示），求马从起点 (x1, y1) 跳到终点 (x2, y2) 最少需要多少步。

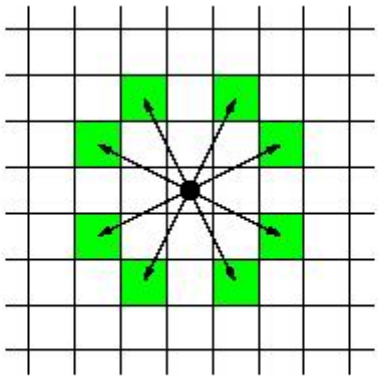


Figure 1: Possible knight moves on the board.

Input

输入一行，包含由空格分隔开的四个整数 x1, y1, x2, y2 （1 <= x1, y1, x2, y2 <= 8），分别表示起点和终点的横纵坐标。

Output

输出一行，包含一个整数，表示从从起点 (x1, y1) 跳到终点 (x2, y2) 所需要的最少步数。

Sample Input

```
5 2 5 4
```

Sample Output

```
2
```

```cpp
/*
A - 马的遍历
 */
#include <bits/stdc++.h>
using namespace std;

using Pair = pair<int, int>;

const vector<Pair> directions = { Pair( 1, 2 ),  Pair( 2, 1 ),   Pair( 2, -1 ),
                                  Pair( 1, -2 ), Pair( -1, -2 ), Pair( -2, -1 ),
                                  Pair( -2, 1 ), Pair( -1, 2 ) };

const int N = 9;
// 该点是否被访问
vector<vector<bool>> visited( N, vector<bool>( N, false ) );
// 到该点要走几步
vector<vector<int>> steps( N, vector<int>( N, -1 ) );

void traverse( Pair &start, Pair &end ) {

    visited[ start.first ][ start.second ] = true;
    steps[ start.first ][ start.second ] = 0;

    queue<Pair> q;
    q.push( start );

    bool found = false;

    while ( !q.empty() ) {
        Pair next = q.front();
        q.pop();
        for ( const auto d : directions ) {
            int x = next.first + d.first;
            int y = next.second + d.second;

            if ( x <= 0 || x >= N || y <= 0 || y >= N ) continue;

            if ( !visited[ x ][ y ] ) {
                visited[ x ][ y ] = true;
                steps[ x ][ y ] = steps[ next.first ][ next.second ] + 1;

                if ( x == end.first && y == end.second ) {
                    found = true;
                    break;
                }

                Pair pre( x, y );
                q.push( pre );
            }
        }
        if ( found ) break;
    }
```

```
}

int main() {
    Pair start, end;
    scanf( "%d%d%d%d", &start.first, &start.second, &end.first, &end.second );

    traverse( start, end );
    printf( "%d\n", steps[ end.first ][ end.second ] );
    return 0;
}

/*
Sample Input
5 2 5 4

Sample Output
2
 */
```

B : 迷宫

Time Limit: `1 Sec` Memory Limit: `128 Mb`

Description

给定一个 N×M 方格的迷宫，迷宫里有T处障碍，障碍处不可通过。给定起点坐标和终点坐标，问: 每个方格最多经过1次，有多少种从起点坐标到终点坐标的方案。在迷宫中移动有上、下、左、右四种方式，每次只能移动一个方格。数据保证起点上没有障碍。

Input

输入的第一行包含三个整数N、M和T（1 <= N, M <= 5, 0 <= T < N * M），其中 N 代表行数，M 代表列数，T 代表障碍总数。

第二行起点坐标SX、SY（1 <= SX <= N, 1 <= SY <= M），终点坐标FX、FY（1 <= FX <= N, 1 <= FY <= M）。

接下来T行，每行为障碍点的坐标X, Y（1 <= X <= N, 1 <= Y <= M）。

Output

输出仅一个整数，表示从起点坐标到终点坐标的方案总数。

Sample Input

```
2 2 1
1 1 2 2
1 2
```

Sample Output

```
1
```

```
/*
B - 迷宫
 */
#include <bits/stdc++.h>
using namespace std;

// #define DEBUG

const int MAXN = 10;

using Pair = pair<int, int>;

vector<Pair> directions( { Pair( 0, 1 ), Pair( 1, 0 ), Pair( 0, -1 ),
                           Pair( -1, 0 ) } );

vector<vector<bool>> blocked( MAXN, vector<bool>( MAXN, false ) );

int N, M, T;
int ans = 0;

void traverse( int x, int y, const Pair &end ) {
    if ( x == end.first && y == end.second ) {
        ++ans;
        return;
    }

    for ( const auto &d : directions ) {
        int xx = x + d.first;
        int yy = y + d.second;

        if ( !blocked[ xx ][ yy ] ) {
            blocked[ x ][ y ] = true;
            traverse( xx, yy, end );
            blocked[ x ][ y ] = false;
        }
    }
}

int main() {
    // 行数，列数，障碍总数
    scanf( "%d%d%d", &N, &M, &T );
    // 起点，终点
    Pair start, end;
    scanf( "%d%d%d%d", &start.first, &start.second, &end.first, &end.second );
    // 障碍的位置
    for ( int i = 0; i < N + 2; i++ ) {
        for ( int j = 0; j < M + 2; j++ ) {
            if ( i == N + 1 || i == 0 || j == M + 1 || j == 0 ) {
                blocked[ i ][ j ] = true;
            }
        }
    }
```

```
    while ( T-- ) {
        int x, y;
        scanf( "%d%d", &x, &y );
        blocked[ x ][ y ] = true;
    }

    traverse( start.first, start.second, end );
    printf( "%d\n", ans );

    return 0;
}

/*
Sample Input
2 2 1
1 1 2 2
1 2

Sample Output
1
 */
```

C：自然数的拆分问题

Time Limit: `1 Sec` Memory Limit: `128 Mb`

Description

任何一个大于 1 的自然数 n，总可以拆分成若干个小于 n 的自然数之和。现在给你一个自然数 n，要求你将 n 拆分成一些数字的和。每个拆分后的序列中的数字从小到大排序。然后你需要输出这些序列，其中字典序小的序列需要优先输出。

Input

输入一个整数 n（1 <= n <= 8），表示需要拆分的自然数。

Output

输出若干个的加法式子。

Sample Input

```
7
```

Sample Output

```
1+1+1+1+1+1+1
1+1+1+1+1+2
1+1+1+1+3
1+1+1+2+2
1+1+1+4
1+1+2+3
1+1+5
1+2+2+2
```

```
1+2+4
1+3+3
1+6
2+2+3
2+5
3+4
```

```cpp
/*
C - 自然数的拆分问题

reference:
https://www.luogu.com.cn/blog/user67087/solution-p2404
 */
#include <bits/stdc++.h>
using namespace std;

vector<int> dp( 16 );

int n;

void display( int t ) {
    for ( int i = 1; i <= t - 1; ++i ) {
        printf( "%d+", dp[ i ] );
    }
    printf( "%d\n", dp[ t ] );
}

void breakdown( int s, int t ) {
    for ( int i = dp[ t - 1 ]; i <= s; ++i ) {
        if ( i < n ) {
            dp[ t ] = i;
            s -= i;

            if ( s == 0 )
                display( t );
            else
                breakdown( s, t + 1 );

            s += i;
        }
    }
}

int main() {
    dp[ 0 ] = 1;
    scanf( "%d", &n );
    breakdown( n, 1 );

    return 0;
}

/*
Sample Input
7

Sample Output
1+1+1+1+1+1+1
```

```
1+1+1+1+1+2
1+1+1+1+3
1+1+1+2+2
1+1+1+4
1+1+2+3
1+1+5
1+2+2+2
1+2+4
1+3+3
1+6
2+2+3
2+5
3+4
 */
```

D : 选数

Time Limit: `1 Sec` Memory Limit: `125 Mb`

Description

已知 n 个整数 $x_1, x_2, \cdots, x_n$，以及 1个整数 k（$k < n$）。从 n 个整数中任选 k 个整数相加，可分别得到一系列的和。例如当 $n = 4$，$k = 3$，4 个整数分别为 3, 7, 12, 19 时，可得全部的组合与它们的和为：

3 + 7 + 12 = 22

3 + 7 + 19 = 29

7 + 12 + 19 = 38

3 + 12 + 19 = 34

现在，要求你计算出和为素数共有多少种。

例如上例，只有一种的和为素数：3 + 7 + 19 = 29。

Input

第一行两个空格隔开的整数 $n, k, (1 \le n \le 20, k < n)$

第二行 n 个整数，分别为 $x_1, x_2, \cdots, x_n$ $(1 \le x_i \le 5 \times 10^6)$。

Output

输出一个整数，表示种类数。

Sample Input

```
4 3
3 7 12 19
```

Sample Output

```
1
```

```
/*
D - 选数

reference:
https://blog.nowcoder.net/n/c4decf47315d45968b47517a6665d909
 */
#include <bits/stdc++.h>
using namespace std;

int n, k;           // n为元素个数，k是参与组合的元素个数
long long ans = 0; // 组合之和为素数的个数

vector<int> arr( 30 );
vector<bool> visited( 30, false );

bool isprime( int num ) {
    for ( int i = 2; i * i <= num; ++i ) {
        if ( num % i == 0 ) return false;
    }

    return true;
}

void dfs( int cnt, int sum, int start_index ) {
    if ( cnt == k ) {
        if ( isprime( sum ) ) ++ans;
        return;
    }

    for ( int i = start_index; i < n; ++i ) {
        if ( !visited[ i ] ) {
            visited[ i ] = true;
            dfs( cnt + 1, sum + arr[ i ], i + 1 );
            visited[ i ] = false;
        }
    }
}

int main() {
    cin >> n >> k;
    for ( int i = 0; i < n; i++ ) {
        cin >> arr[ i ];
    }
    dfs( 0, 0, 0 );
    cout << ans << endl;
    return 0;
}

/*
Sample Input
4 3
3 7 12 19
```

```
Sample Output
1
 */
```

E : 好奇怪的游戏

Description

这个游戏类似象棋，但是只有黑白马各一匹，在点 $x1,y1$ 和 $x2,y2$ 上。它们得从点 $x1,y1$ 和 $x2,y2$ 走到 (1,1) 。这个游戏与普通象棋不同的地方是：马可以走“日”，也可以像象走“田”。现在想知道两匹马到 (1,1) 的最少步数，你能解决这个问题么？

Input

第1行：两个整数$x1,y1$

第2行：两个整数$x2,y2$

($1 \le x1,y1,x2,y2 \le 20$)

Output

第1行：黑马到(1,1)的步数

第2行：白马到(1,1)的步数

假设棋盘为 `20*20`

Sample Input

```
12 16
18 10
```

Sample Output

```
8
9
```

```
/*
E - 好奇怪的游戏
 */
#include <bits/stdc++.h>
using namespace std;

using Pair = pair<int, int>;
using PairArray = vector<Pair>;
using IntArray = vector<int>;
using IntMatrix = vector<IntArray>;

const int SIZE = 20;

const PairArray directions( { Pair( 1, 2 ), Pair( 2, 1 ), Pair( 2, -1 ),
                              Pair( 1, -2 ), Pair( -1, -2 ), Pair( -2, -1 ),
```

```
                                Pair( -2, 1 ), Pair( -1, 2 ), Pair( 2, 2 ),
                                Pair( 2, -2 ), Pair( -2, -2 ), Pair( -2, 2 ) } );
IntMatrix steps;

void dfs( int x, int y, Pair &end, int step ) {
    steps[ x ][ y ] = step;

    for ( const auto &d : directions ) {
        int xx = x + d.first;
        int yy = y + d.second;

        if ( ( xx >= 1 && xx <= SIZE && yy >= 1 && yy <= SIZE ) &&
             ( steps[ xx ][ yy ] == -1 || steps[ xx ][ yy ] > step + 1 ) ) {
            dfs( xx, yy, end, step + 1 );

            if ( xx == end.first && yy == end.second ) break;
        }
    }
}

int main() {
    Pair start;
    Pair end( 1, 1 );

    int rounds = 2;
    while ( rounds-- ) {
        cin >> start.first >> start.second;
        steps = IntMatrix( SIZE + 10, IntArray( SIZE + 10, -1 ) );
        dfs( start.first, start.second, end, 0 );
        cout << steps[ end.first ][ end.second ] << endl;
    }

    return 0;
}

/*
Sample Input
12 16
18 10

Sample Output
8
9
 */
```

1039 经典回溯

A : N皇后问题

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

规定当两个皇后出现在同一行，同一列，或者同一条对角线上时，它们会互相攻击。 现在要在一个 N×N 的棋盘上放置 N 个皇后，使其不能互相攻击，即任意两个皇后都不能处于同一行、同一列或同一斜线上，请问有多少种放置方法。

Input

包含多组测试数据。 每组数据输入一行，仅包含一个整数 N （1 <= N <= 13）。

Output

每组数据输出一行，包含一个整数，表示放置方案总数。

Sample Input

```
8
6
13
```

Sample Output

```
92
4
73712
```

```cpp
/*
A - N皇后问题

reference:
https://leetcode-cn.com/problems/n-queens/solution/nhuang-hou-bu-yong-ha-xi-biao-shi-yong-b-d2vj/
 */
#include <bits/stdc++.h>
using namespace std;

int ans;

void dfs( int n, int row, vector<bool> &cols, vector<bool> &diags,
          vector<bool> &antidiags ) {
    if ( n == row ) {
        ++ans;
    } else {
        for ( int i = 0; i < n; ++i ) {
            if ( cols[ i ] | diags[ i + row ] | antidiags[ n - 1 - row + i ] )
                continue;

            cols[ i ] = diags[ i + row ] = antidiags[ n - 1 - row + i ] = true;
            dfs( n, row + 1, cols, diags, antidiags );
            cols[ i ] = diags[ i + row ] = antidiags[ n - 1 - row + i ] = false;
        }
    }
}

int main() {
```

```
    int N;
    while ( scanf( "%d", &N ) != EOF ) {
        ans = 0;

        vector<bool> cols( N, false );
        vector<bool> diags( 2 * N - 1, false );
        vector<bool> antidiags( 2 * N - 1, false );

        dfs( N, 0, cols, diags, antidiags );

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

    return 0;
}

/*
Sample Input
8
6
13

Sample Output
92
4
73712
 */
```

B : 吃奶酪

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

房间里放着 *n* 块奶酪。一只小老鼠要把它们都吃掉，问至少要跑多少距离？老鼠一开始在 (0,0)点处。

Input

第一行有一个整数，表示奶酪的数量 *n*。

第 2 到第 (*n* + 1) 行，每行两个实数，第 (*i* + 1) 行的实数分别表示第 *i* 块奶酪的横纵坐标 x_i, y_i。

Output

输出一行一个实数，表示要跑的最少距离，保留 2 位小数。

Sample Input

```
4
1 1
1 -1
-1 1
-1 -1
```

Sample Output

```
7.41
```

Hint

数据规模与约定

对于全部的测试点，保证 $1 \leq n \leq 15, |x_i|, |y_i| \leq 200$，小数点后最多有 3位数字。

提示

对于两个点 $(x_1, y_1), (x_2, y_2)$两点之间的距离公式为$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

```
/*
B - 吃奶酪

reference:
https://blog.51cto.com/u_3044148/3349412

状态压缩：
一维：代表已经走过的节点有哪些，是一个二进制转为十进制的数字，最终表现为十进制.
    每一位代表一个奶酪，比如一共有3个奶酪，f[(101)二进制]
=f[5]，表示1号和3号奶酪吃完了，2号没有吃  二维：代表当前的出发点
值：f[i][j]表示如果以前计算过在以第j个位置出发，在前面已经完成i这种二进制表示法的节点完成情况
下，最短的距离是多少
 */
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 16;

int InputNum;
double ans = 0x3f3f3f3f;
double min_dist = 0x3f3f3f3f;

vector<double> X( MAXN );
vector<double> Y( MAXN );
vector<bool> visited( MAXN );
double dp[ 1 << MAXN ][ MAXN ];
vector<vector<double>> Distance( MAXN, vector<double>( MAXN ) );

double get_distance( double x1, double y1, double x2, double y2 ) {
    return sqrt( ( x1 - x2 ) * ( x1 - x2 ) + ( y1 - y2 ) * ( y1 - y2 ) );
}

void dfs( int pos, int step, int state, double len ) {

    // 代价函数
    double mayBestAns = len + min_dist * ( InputNum - step );
    // 用代价函数做一次剪枝
    if ( mayBestAns > ans ) return;

    if ( step == InputNum ) {
        ans = std::min( ans, len );
        return;
```

```
    }

    for ( int i = 1; i <= InputNum; ++i ) {
        if ( visited[ i ] ) continue;

        // 下一状态
        int next_state = state + ( 1 << ( i - 1 ) );
        // 用压缩状态剪枝
        // 如果尝试过且更好就不必再去了
        if ( dp[ next_state ][ i ] &&
             dp[ next_state ][ i ] <= len + Distance[ pos ][ i ] )
            continue;

        visited[ i ] = true;

        dp[ next_state ][ i ] = len + Distance[ pos ][ i ];
        dfs( i, step + 1, next_state, dp[ next_state ][ i ] );

        visited[ i ] = false;
    }
}

int main() {
    scanf( "%d", &InputNum );

    for ( int i = 1; i <= InputNum; ++i ) {
        scanf( "%lf%lf", &X[ i ], &Y[ i ] );

        for ( int j = 0; j < i; ++j ) {
            Distance[ i ][ j ] = Distance[ j ][ i ] =
                get_distance( X[ i ], Y[ i ], X[ j ], Y[ j ] );
            min_dist = std::min( min_dist, Distance[ i ][ j ] );
        }
    }

    dfs( 0, 0, 0, 0 );

    printf( "%.2f\n", ans );

    return 0;
}

/*
Sample Input
4
1 1
1 -1
-1 1
-1 -1

Sample Output
7.41
 */
```

C : RGB Coloring 2

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

We have a simple undirected graph with *N* vertices and *M* edges. The vertices are numbered 1 through *N* and the edges are numbered 1 through *M*. Edge *i* connects Vertex A_i and Vertex B_i. Find the number of ways to paint each vertex in this graph red, green, or blue so that the following condition is satisfied:

- two vertices directly connected by an edge are always painted in different colors.

Here, it is not mandatory to use all the colors.

Input

Input is given from Standard Input in the following format:

$N\ M$

$A_1\ B_1$

$A_2\ B_2$

.

$A_M\ B_M$

Output

Print the answer.

Sample Input

```
3 3
1 2
2 3
3 1
```

Sample Output

```
6
```

Hint

数据范围

- $1 \leq N \leq 20$
- $0 \leq M \leq \frac{N(N-1)}{2}$
- $1 \leq A_i \leq N$
- $1 \leq B_i \leq N$
- The given graph is simple (that is, has no multi-edges and no self-loops).
- Note that the graph may not be connected.

```
/*
C - RGB Coloring 2
 */
```

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

const int maxn = 100;

vector<vector<bool>> nbrs( maxn, vector<bool>( maxn, false ) );
vector<int> colors( maxn, 0 );
vector<bool> visited( maxn, false );
vector<int> continuous( maxn, 0 );

int N, M;
int solution_cnt, continuous_cnt;

void get_continuous( int id ) {
    visited[ id ] = true;
    continuous[ continuous_cnt++ ] = id;

    for ( int i = 1; i <= N; ++i ) {
        if ( !visited[ i ] && nbrs[ i ][ id ] ) {
            get_continuous( i );
        }
    }
}

bool isvalid( int id, int color ) {
    for ( int i = 1; i <= M; ++i ) {
        if ( nbrs[ id ][ i ] && color == colors[ i ] ) {
            return false;
        }
    }
    return true;
}

void dfs( int id ) {
    if ( id == continuous_cnt ) {
        ++solution_cnt;
        return;
    }

    for ( int i = 1; i <= 3; ++i ) {
        if ( !isvalid( continuous[ id ], i ) ) continue;

        colors[ continuous[ id ] ] = i;
        dfs( id + 1 );
        colors[ continuous[ id ] ] = 0;
    }
}

int main() {
    cin >> N >> M;

    for ( int i = 0; i < M; ++i ) {
        int a, b;
        cin >> a >> b;
        nbrs[ a ][ b ] = nbrs[ b ][ a ] = true;
    }

    long long ans = 1;
```

```
    for ( int i = 1; i <= N; ++i ) {
        if ( visited[ i ] ) continue;

        solution_cnt = continuous_cnt = 0;
        get_continuous( i );
        dfs( 0 );
        ans *= solution_cnt;
    }
    cout << ans << endl;
    return 0;
}

/*
Sample Input
3 3
1 2
2 3
3 1

Sample Output
6
 */
```

1043 线性规划理论

A : 防守战线

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

战线可以看作一个长度为 n 的序列，现在需要在这个序列上建塔来防守敌兵，在序列第 *i* 号位置上建一座塔的花费为 C_i，且一个位置可以建任意多的塔，费用累加计算。有 *m* 个区间 $[L_1, R_1],, [L_2, R_2]$ ..., $[L_m, R_m]$，在第 *i* 个区间的范围内要建至少 D_i 座塔，求最少花费。

Input

第一行为两个数 n, m ($n \leq 1000, m \leq 10000$)，分别表示序列长度及区间个数

第二行有 n 个数，描述序列 C ($C_i \leq 10000$)

接下来 m 行，每行有三个数 L_i, R_i, D_i，描述一个区间 ($1 \leq L_i \leq R_i \leq n$, $D_i \leq 10000$)

Output

一个整数，代表最少花费

Sample Input

```
5 3
1 5 6 3 4
2 3 1
1 5 4
3 5 2
```

Sample Output

```
11
```

Hint

位置 1 建 2 个塔，位置 3 建 1个塔，位置 4 建 1 个塔，共花费 1 * 2 + 6 + 3 = 11 。

```cpp
/*
A - 防守战线

reference:
https://kirainmoe.com/blog/post/simplex-algorithm/
https://www.codetd.com/article/4727804
 */
#include <bits/stdc++.h>
using namespace std;

namespace simplex {

using Array = vector<double>;
using Matrix = vector<Array>;

const double eps = 1e-8, inf = (double) ( 1ll << 60 );
Matrix A;

class Solver {
  private:
    int m;
    int n;

  public:
    Solver( int _m, int _n )
        : m( _m )
        , n( _n ) {}

    void input_matrix( const Matrix &a, const Array &b, const Array &c ) {
        for ( int i = 1; i <= n; ++i ) {
            A[ 0 ][ i ] = c[ i - 1 ];
        }
        for ( int i = 1; i <= m; ++i ) {
            for ( int j = 1; j <= n; ++j ) {
                A[ i ][ j ] = a[ i - 1 ][ j - 1 ];
            }
            A[ i ][ 0 ] = b[ i - 1 ]; // input b
        }
    }

    void display() const {
        cout << "m=" << m << ", n=" << n << endl;

        for ( int i = 0; i <= m; ++i ) {
            for ( int j = 0; j <= n; ++j ) {
                cout << "\t" << A[ i ][ j ];
            }
            cout << endl;
```

```
        }
    }

    void pivot( int l, int e ) {
        double tmp = A[ l ][ e ];
        A[ l ][ e ] = 1.0;
        vector<int> tmp_arr;
        for ( int j = 0; j <= n; ++j ) {
            if ( std::fabs( A[ l ][ j ] ) > 0 ) {
                A[ l ][ j ] /= tmp;
                tmp_arr.push_back( j );
            }
        }
        int cnt = tmp_arr.size();
        for ( int i = 0; i <= m; i++ ) {
            if ( i != l && std::fabs( A[ i ][ e ] ) > eps ) {
                tmp = A[ i ][ e ], A[ i ][ e ] = 0;
                for ( int j = 0; j < cnt; ++j ) {
                    A[ i ][ tmp_arr[ j ] ] -= A[ l ][ tmp_arr[ j ] ] * tmp;
                }
            }
        }
    }

    bool simplex( bool verbose = false ) {
        while ( true ) {
            int l = 0, e = 0;
            double minv = inf;
            for ( int j = 1; j <= n; ++j ) {
                if ( A[ 0 ][ j ] > eps ) {
                    e = j;
                    break;
                }
            }
            if ( !e ) break;
            for ( int i = 1; i <= m; ++i ) {
                if ( A[ i ][ e ] > eps && A[ i ][ 0 ] / A[ i ][ e ] < minv ) {
                    minv = A[ i ][ 0 ] / A[ i ][ e ];
                    l = i;
                }
            }
            if ( !l ) {
                // printf( "Unbounded\n" );
                return false;
            }
            pivot( l, e );

            if ( verbose ) display();
        }
        return true;
    }

    double get_result() const { return -A[ 0 ][ 0 ]; }

    void standardize() {
        for ( int i = 0; i < n; ++i )
            A[ 0 ].push_back( 0 );
        for ( int i = 1; i <= m; ++i ) {
```

```
            for ( int j = 1; j <= m; ++j ) {
                if ( i == j ) {
                    A[ i ].push_back( 1 );
                } else {
                    A[ i ].push_back( 0 );
                }
            }
        }
        n += m;
    }
};

} // namespace simplex

int main() {
    using namespace simplex;

    int m, n;
    cin >> m >> n;

    A = Matrix( m + 1, Array( n + 1 ) );

    for ( int i = 1; i <= m; ++i ) {
        cin >> A[ i ][ 0 ];
    }

    int l, r;
    for ( int i = 1; i <= n; ++i ) {
        cin >> l >> r >> A[ 0 ][ i ];
        for ( int j = l; j <= r; ++j )
            A[ j ][ i ]++;
    }

    Solver s( m, n );
    // s.standardize();
    // s.display();

    s.simplex( false );
    // s.display();

    cout << int( s.get_result() ) << endl;

    return 0;
}

/*
Sample Input
5 3
1 5 6 3 4
2 3 1
1 5 4
3 5 2

Sample Output
11
*/
```

B : 序列

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

给定一个长度为 n 的正整数序列 c_i，求一个子序列，满足原序列中任意长度为 m 的子串中被选出的元素个数不超过 k，同时使得子序列的元素之和最大。

子序列和子串的定义如下：

子串：从原序列中截取的任意长度的连续字符即为该序列的子串

子序列：从原序列中选取任意个字符（不一定连续）按原序列中的顺序组成的新序列称为该序列的子序列

如：原序列为 $abcde$，则 bcd 为原序列的一个子串同时也是一个子序列，bce 为原序列的一个子序列但不为子串。

Input

第一行为三个数 n, m, k ($n \leq 1000, k \leq 100, m \leq 100$)

接下来 n 行，每行一个字符串表示 c_i($c_i \leq 2000$)

Output

输出一个正整数，表示子序列的元素之和

Sample Input

```
10 5 3
4
4
4
6
6
6
6
6
4
4
```

Sample Output

```
30
```

```cpp
/*
B - 序列
 */
#include <bits/stdc++.h>
using namespace std;

namespace simplex {

using Array = vector<double>;
```

```
using Matrix = vector<Array>;

const double eps = 1e-8, inf = (double) ( 1ll << 60 );
Matrix A;

class Solver {
  private:
    int m;
    int n;

  public:
    Solver( int _m, int _n )
        : m( _m )
        , n( _n ) {}

    void input_matrix( const Matrix &a, const Array &b, const Array &c ) {
        for ( int i = 1; i <= n; ++i ) {
            A[ 0 ][ i ] = c[ i - 1 ];
        }
        for ( int i = 1; i <= m; ++i ) {
            for ( int j = 1; j <= n; ++j ) {
                A[ i ][ j ] = a[ i - 1 ][ j - 1 ];
            }
            A[ i ][ 0 ] = b[ i - 1 ]; // input b
        }
    }

    void display() const {
        cout << "m=" << m << ", n=" << n << endl;

        for ( int i = 0; i <= m; ++i ) {
            for ( int j = 0; j <= n; ++j ) {
                cout << "\t" << A[ i ][ j ];
            }
            cout << endl;
        }
    }

    void pivot( int l, int e ) {
        double tmp = A[ l ][ e ];
        A[ l ][ e ] = 1.0;
        vector<int> tmp_arr;
        for ( int j = 0; j <= n; ++j ) {
            if ( std::fabs( A[ l ][ j ] ) > 0 ) {
                A[ l ][ j ] /= tmp;
                tmp_arr.push_back( j );
            }
        }
        int cnt = tmp_arr.size();
        for ( int i = 0; i <= m; i++ ) {
            if ( i != l && std::fabs( A[ i ][ e ] ) > eps ) {
                tmp = A[ i ][ e ], A[ i ][ e ] = 0;
                for ( int j = 0; j < cnt; ++j ) {
                    A[ i ][ tmp_arr[ j ] ] -= A[ l ][ tmp_arr[ j ] ] * tmp;
                }
            }
        }
    }
```

```
    bool simplex( bool verbose = false ) {
        while ( true ) {
            int l = 0, e = 0;
            double minv = inf;
            for ( int j = 1; j <= n; ++j ) {
                if ( A[ 0 ][ j ] > eps ) {
                    e = j;
                    break;
                }
            }
            if ( !e ) break;
            for ( int i = 1; i <= m; ++i ) {
                if ( A[ i ][ e ] > eps && A[ i ][ 0 ] / A[ i ][ e ] < minv ) {
                    minv = A[ i ][ 0 ] / A[ i ][ e ];
                    l = i;
                }
            }
            if ( !l ) {
                // printf( "Unbounded\n" );
                return false;
            }
            pivot( l, e );

            if ( verbose ) display();
        }
        return true;
    }

    double get_result() const { return -A[ 0 ][ 0 ]; }

    void standardize() {
        for ( int i = 0; i < n; ++i )
            A[ 0 ].push_back( 0 );
        for ( int i = 1; i <= m; ++i ) {
            for ( int j = 1; j <= m; ++j ) {
                if ( i == j ) {
                    A[ i ].push_back( 1 );
                } else {
                    A[ i ].push_back( 0 );
                }
            }
        }
        n += m;
    }
};

} // namespace simplex

int main() {
    using namespace simplex;

    int n, m, k;
    cin >> n >> m >> k;

    int bn = 2 * n - m + 1, cn = n;
    int row = n - m + 1;
```

```
    A = Matrix( bn + 1, Array( cn + 1 ) );
    for ( int i = 1; i <= cn; ++i )
        cin >> A[ 0 ][ i ];

    for ( int i = 1; i <= row; ++i ) {
        A[ i ][ 0 ] = k;
        for ( int j = i; j < i + m; ++j )
            A[ i ][ j ] = 1;
    }

    for ( int i = row + 1; i <= bn; ++i ) {
        A[ i ][ 0 ] = 1;
        A[ i ][ i - row ] = 1;
    }

    Solver s( bn, cn );
    // s.standardize();
    // s.display();
    s.simplex( false );
    // s.display();
    printf( "%d\n", int( s.get_result() + 0.5 ) );
    return 0;
}

/*
Sample Input
10 5 3
4
4
4
6
6
6
6
6
4
4

Sample Output
30
*/
```

1047 线性规划运用

A : 志愿者招募

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

奥运将至，布布需要为奥运项目招募一批短期志愿者。经过估算，这个项目需要 *n* 天才能完成，其中第 *i* 天至少需要 a_i 个志愿者。布布通过了解得知，一共有 *m* 类志愿者可以抬募。其中第 *i* 类可以从第 s_i 天工作到t_i天，招募费用是每人c_i 元。布布希望用尽量少的费用招足够多的志愿者，请你帮他设计一种最优的招募方案。

Input

第一行包含两个整数 n, m, 表示完成项目的天数和可以招募的志愿者的种类数。($1 \le n \le 1000, 1 \le m \le 10000$)

接下来的一行中包含 n 个非负整数，表示各天至少需要的志愿者人数。

接下来的 m 行中每行包含三个整数 s_i, t_i, c_i, 含义如上文所述。为了方便起见，我们可以认为每类志愿者的数量都是无限多的，同时保证一定存在最优方案。($1 \le s_i \le t_i \le n$, $c_i \le 2^{31} - 1$)

Output

仅包含一个整数，表示你所设计的最优方案的总费用。

Sample Input

```
3 3
2 3 4
1 2 2
2 3 5
3 3 2
```

Sample Output

```
14
```

```cpp
/*
A - 志愿者招募
 */
#include <bits/stdc++.h>
using namespace std;

namespace simplex {

using Array = vector<double>;
using Matrix = vector<Array>;

const double eps = 1e-8, inf = (double) ( 1ll << 60 );
// 系数矩阵，第一行为c，第一列为b
Matrix A;

class Solver {
  private:
    int m;
    int n;

  public:
    Solver( int _m, int _n )
        : m( _m )
        , n( _n ) {}

    void input_matrix( const Matrix &a, const Array &b, const Array &c ) {
        for ( int i = 1; i <= n; ++i ) {
            A[ 0 ][ i ] = c[ i - 1 ];
        }
```

```
        for ( int i = 1; i <= m; ++i ) {
            for ( int j = 1; j <= n; ++j ) {
                A[ i ][ j ] = a[ i - 1 ][ j - 1 ];
            }
            A[ i ][ 0 ] = b[ i - 1 ]; // input b
        }
    }

    void display() const {
        cout << "m=" << m << ", n=" << n << endl;

        for ( int i = 0; i <= m; ++i ) {
            for ( int j = 0; j <= n; ++j ) {
                cout << "\t" << A[ i ][ j ];
            }
            cout << endl;
        }
    }

    void pivot( int l, int e ) {
        double tmp = A[ l ][ e ];
        A[ l ][ e ] = 1.0;
        vector<int> tmp_arr;
        for ( int j = 0; j <= n; ++j ) {
            if ( std::fabs( A[ l ][ j ] ) > 0 ) {
                A[ l ][ j ] /= tmp;
                tmp_arr.push_back( j );
            }
        }
        int cnt = tmp_arr.size();
        for ( int i = 0; i <= m; i++ ) {
            if ( i != l && std::fabs( A[ i ][ e ] ) > eps ) {
                tmp = A[ i ][ e ], A[ i ][ e ] = 0;
                for ( int j = 0; j < cnt; ++j ) {
                    A[ i ][ tmp_arr[ j ] ] -= A[ l ][ tmp_arr[ j ] ] * tmp;
                }
            }
        }
    }

    bool simplex( bool verbose = false ) {
        while ( true ) {
            int l = 0, e = 0;
            double minv = inf;
            for ( int j = 1; j <= n; ++j ) {
                if ( A[ 0 ][ j ] > eps ) {
                    e = j;
                    break;
                }
            }
            if ( !e ) break;
            for ( int i = 1; i <= m; ++i ) {
                if ( A[ i ][ e ] > eps && A[ i ][ 0 ] / A[ i ][ e ] < minv ) {
                    minv = A[ i ][ 0 ] / A[ i ][ e ];
                    l = i;
                }
            }
            if ( !l ) {
```

```
                // printf( "Unbounded\n" );
                return false;
            }
            pivot( l, e );

            if ( verbose ) display();
        }
        return true;
    }

    double get_result() const { return -A[ 0 ][ 0 ]; }

    void standardize() {
        for ( int i = 0; i < n; ++i )
            A[ 0 ].push_back( 0 );
        for ( int i = 1; i <= m; ++i ) {
            for ( int j = 1; j <= m; ++j ) {
                if ( i == j ) {
                    A[ i ].push_back( 1 );
                } else {
                    A[ i ].push_back( 0 );
                }
            }
        }
        n += m;
    }
};

} // namespace simplex

int main() {
    using namespace simplex;

    int n, m;
    cin >> n >> m;

    A = Matrix( m + 1, Array( n + 1 ) );
    for ( int i = 1; i <= n; ++i )
        cin >> A[ 0 ][ i ];
    for ( int i = 1; i <= m; ++i ) {
        int s, t;
        cin >> s >> t >> A[ i ][ 0 ];
        for ( int j = s; j <= t; ++j )
            A[ i ][ j ] = 1;
    }

    Solver s( m, n );
    // s.display();
    s.simplex( false );
    // s.display();

    cout << int( s.get_result() + 0.5 ) << endl;
    return 0;
}

/*
Sample Input
3 3
```

```
2 3 4
1 2 2
2 3 5
3 3 2

Sample Output
14
*/
```

B : 选数

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

有 3N 个数 (N ≤ 200)，你需要选出一些数，首先保证任意长度为N的区间中选出的数的个数 ≤ K个 (K ≤ 10)，其次要保证选出的数个数最大

Input

第一行包括两个整数 N, K

第二行有 3N 个整数

Output

输出一行包括一个整数表示选出数的总和

Sample Input

```
5 3
14 21 9 30 11 8 1 20 29 23 17 27 7 8 35
```

Sample Output

```
195
```

```cpp
/*
B - 选数
 */
#include <bits/stdc++.h>
using namespace std;

namespace simplex {

using Array = vector<double>;
using Matrix = vector<Array>;

const double eps = 1e-8, inf = (double) ( 1ll << 60 );
// 系数矩阵，第一行为c，第一列为b
Matrix A;

class Solver {
  private:
```

```
    int m;
    int n;

  public:
    Solver( int _m, int _n )
        : m( _m )
        , n( _n ) {}

    void input_matrix( const Matrix &a, const Array &b, const Array &c ) {
        for ( int i = 1; i <= n; ++i ) {
            A[ 0 ][ i ] = c[ i - 1 ];
        }
        for ( int i = 1; i <= m; ++i ) {
            for ( int j = 1; j <= n; ++j ) {
                A[ i ][ j ] = a[ i - 1 ][ j - 1 ];
            }
            A[ i ][ 0 ] = b[ i - 1 ]; // input b
        }
    }

    void display() const {
        cout << "m=" << m << ", n=" << n << endl;

        for ( int i = 0; i <= m; ++i ) {
            for ( int j = 0; j <= n; ++j ) {
                cout << "\t" << A[ i ][ j ];
            }
            cout << endl;
        }
    }

    void pivot( int l, int e ) {
        double tmp = A[ l ][ e ];
        A[ l ][ e ] = 1.0;
        vector<int> tmp_arr;
        for ( int j = 0; j <= n; ++j ) {
            if ( std::fabs( A[ l ][ j ] ) > 0 ) {
                A[ l ][ j ] /= tmp;
                tmp_arr.push_back( j );
            }
        }
        int cnt = tmp_arr.size();
        for ( int i = 0; i <= m; i++ ) {
            if ( i != l && std::fabs( A[ i ][ e ] ) > eps ) {
                tmp = A[ i ][ e ], A[ i ][ e ] = 0;
                for ( int j = 0; j < cnt; ++j ) {
                    A[ i ][ tmp_arr[ j ] ] -= A[ l ][ tmp_arr[ j ] ] * tmp;
                }
            }
        }
    }

    bool simplex( bool verbose = false ) {
        while ( true ) {
            int l = 0, e = 0;
            double minv = inf;
            for ( int j = 1; j <= n; ++j ) {
                if ( A[ 0 ][ j ] > eps ) {
```

```
                    e = j;
                    break;
                }
            }
            if ( !e ) break;
            for ( int i = 1; i <= m; ++i ) {
                if ( A[ i ][ e ] > eps && A[ i ][ 0 ] / A[ i ][ e ] < minv ) {
                    minv = A[ i ][ 0 ] / A[ i ][ e ];
                    l = i;
                }
            }
            if ( !l ) {
                // printf( "Unbounded\n" );
                return false;
            }
            pivot( l, e );

            if ( verbose ) display();
        }
        return true;
    }

    double get_result() const { return -A[ 0 ][ 0 ]; }

    void standardize() {
        for ( int i = 0; i < n; ++i )
            A[ 0 ].push_back( 0 );
        for ( int i = 1; i <= m; ++i ) {
            for ( int j = 1; j <= m; ++j ) {
                if ( i == j ) {
                    A[ i ].push_back( 1 );
                } else {
                    A[ i ].push_back( 0 );
                }
            }
        }
        n += m;
    }
};

} // namespace simplex

int main() {
    using namespace simplex;

    int N, K;
    cin >> N >> K;

    int row = 2 * N + 1;
    int bn = 5 * N + 1, cn = 3 * N;
    A = Matrix( bn + 1, Array( cn + 1 ) );

    for ( int i = 1; i <= cn; ++i )
        cin >> A[ 0 ][ i ];
    for ( int i = 1; i <= row; ++i ) {
        A[ i ][ 0 ] = K;
        for ( int j = i; j < i + N; ++j ) {
            A[ i ][ j ] = 1;
```

```
        }
    }
    for ( int i = row + 1; i <= row + cn; ++i ) {
        A[ i ][ 0 ] = 1;
        A[ i ][ i - row ] = 1;
    }

    Solver s( bn, cn );
    // s.display();

    s.simplex();
    // s.display();

    cout << int( s.get_result() + 0.5 ) << endl;

    return 0;
}

/*
Sample Input
5 3
14 21 9 30 11 8 1 20 29 23 17 27 7 8 35

Sample Output
195
*/
```

C : 招募员工

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

一家 24 小时营业的超市需要雇佣一些员工来满足超市的日常经营需求，为此超市经理特地请你来协助他。

超市每天的不同时间段需要不同数量的员工（例如，下午人流量多，需要的员工也多，深夜人流量少，需要的员工少）来为顾客提供服务。超市经理提供了一天中每一个小时时段所需的最少员工数（第一个时间段为 0 点 - 1 点，共 24 个时间段，注意，这些时间段内需要的员工数每天都是相同的）。

现在有 n 个人可以雇佣，每个人有固定的工作时间段，每天从指定时间 t_i 点钟开始，连续工作 8 个小时。

请你求出在满足人员数量要求的前提下，最少需要雇佣的员工数。

Input

第一行为一个正整数 $T(T \le 20)$，代表测试样例数

第二行为 24 个整数，依次代表 24 个时间段内所需的最少员工数 (每个时间段不超过 1000)

第三行为一个整数 $n(0 \le n \le 1000)$，代表可雇佣人数

接下来的 n 行每行一个整数 t_i $(0 \le t_i \le 23)$，代表第 i 个人每天的工作开始时间

Output

对于每一组测试样例，若该测试样例有解，输出最少需要雇佣的员工数，否则输出 “No Solution” (不带引号)。

Sample Input

```
1
1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
5
0
23
22
1
10
```

Sample Output

```
1
```

```
/*
C - 招募员工

reference:
https://blog.csdn.net/consciousman/article/details/53812818
 */
#include <bits/stdc++.h>
using namespace std;

const int N = 30, M = 1010;
const int INF = 0x3f3f3f3f;

struct edge {
    int v, d, next;
    edge( int v, int d, int n )
        : v( v )
        , d( d )
        , next( n ) {}
    edge() {}
} ed[ M ];

int k;
vector<int> head, d, R, tim;
vector<bool> visited;
queue<int> q;

void init() {
    k = 0;
    head = vector<int>( N, -1 );
    d = vector<int>( N, -INF );
    visited = vector<bool>( N, false );

    while ( !q.empty() )
        q.pop();
}
```

```
void add( int u, int v, int d ) {
    ed[ k ] = edge( v, d, head[ u ] );
    head[ u ] = k++;
}

int cal( int m ) {
    q.push( 0 );
    d[ 0 ] = 0;
    while ( !q.empty() ) {
        int x = q.front();
        q.pop();

        visited[ x ] = false;
        if ( x == 24 && d[ x ] > m ) return 0;
        for ( int i = head[ x ]; i != -1; i = ed[ i ].next ) {
            int t = ed[ i ].v;
            if ( d[ t ] < d[ x ] + ed[ i ].d ) {
                d[ t ] = d[ x ] + ed[ i ].d;
                if ( !visited[ t ] ) {
                    visited[ t ] = true;
                    q.push( t );
                }
            }
        }
    }
    return d[ 24 ] == m ? 1 : 0;
}

int main() {
    int t;
    cin >> t;
    while ( t-- ) {
        R = vector<int>( N );
        for ( int i = 0; i < 24; i++ )
            cin >> R[ i ];

        int n;
        cin >> n;

        tim = vector<int>( N );
        for ( int i = 0; i < n; i++ ) {
            int tmp;
            cin >> tmp;
            tim[ tmp ]++;
        }
        int r = n + 1, l = -1;
        while ( r - l > 1 ) {
            int m = ( r + l ) / 2;
            init();
            for ( int i = 0; i <= 23; i++ )
                add( i + 1, i, -tim[ i ] ), add( i, i + 1, 0 );
            for ( int i = 7; i <= 23; i++ )
                add( i - 7, i + 1, R[ i ] );

            add( 0, 24, m );
            add( 24, 0, -m );
```

```
            for ( int i = 0; i < 7; i++ )
                add( i + 17, i + 1, R[ i ] - m );
            cal( m ) ? r = m : l = m;
        }
        l >= n ? puts( "No Solution" ) : printf( "%d\n", r );
    }
    return 0;
}

/*
Sample Input
1
1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
5
0
23
22
1
10

Sample Output
1
*/
```

D : 餐巾计划问题

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

一个餐厅在相继的 *N* 天里,每天需用的餐巾数不尽相同。假设第 *i* 天需要 r_i 块餐巾(i = 1, 2, ..., N)。餐厅可以购买新的餐巾,每块餐巾的费用为 p*p* 分;或者把旧餐巾送到快洗部,洗一块需 m 天,其费用为 f 分;或者送到慢洗部,洗一块需 *n* 天 (*n* > *m*), 其费用为 *s* 分 (*s* < *f*)。

每天结束时,餐厅必须决定将多少块脏的餐巾送到快洗部,多少块餐巾送到慢洗部,以及多少块保存起来延期送洗。但是每天洗好的餐巾和购买的新餐巾数之和,要满足当天的需求量。

试设计一个算法为餐厅合理地安排好 *N* 天中餐巾使用计划,使总的花费最小。编程找出一个最佳餐巾使用计划。

Input

由标准输入提供输入数据。文件第 1 行有 1 个正整数 *N*，代表要安排餐巾使用计划的天数。

接下来的一行是餐厅在相继的 *N* 天里,每天需用的餐巾数。

最后一行包含5个正整数 *p*, *m*, *f*, *n*, *s*。*p* 是每块新餐巾的费用; *m* 是快洗部洗一块餐巾需用天数; *f* 是快洗部洗一块餐巾需要的费用; *n* 是慢洗部洗一块餐巾需用天数; *s* 是慢洗部洗一块餐巾需要的费用。

Output

将餐厅在相继的 N 天里使用餐巾的最小总花费输出

Sample Input

```
3
1 7 5
11 2 2 3 1
```

Sample Output

```
134
```

Hint

N ≤ 2000

ri ≤ 10000000

p,n,f,m,s ≤ 10000

时限4s

```
/*
D - 餐巾计划问题

reference:
https://blog.51cto.com/u_15180869/2732667
 */
#include <bits/stdc++.h>
using namespace std;

using LL = long long;

const LL maxn = 100005, inf = 0x3f3f3f3f3f3f3f3f;

struct edge {
    LL to, next, cost, capacity;

    edge( LL t, LL n, LL c, LL v )
        : to( t )
        , next( n )
        , cost( c )
        , capacity( v ) {}
    edge() {}
} e[ maxn << 1 ];

LL N, n, p, m, f, s, cnt, ans;
vector<LL> nexts( maxn, -1 ), dist;
vector<bool> visited;

inline void add( LL from, LL to, LL capacity, LL cost ) {
    e[ cnt ] = edge( to, nexts[ from ], cost, capacity );
    nexts[ from ] = cnt++;
    // 反向边
    e[ cnt ] = edge( from, nexts[ to ], -cost, 0 );
    nexts[ to ] = cnt++;
}
```

```cpp
inline bool spfa() {
    deque<LL> q;
    dist = vector<LL>( maxn, -1 );
    visited = vector<bool>( maxn, false );
    // 汇点
    dist[ N * 2 + 1 ] = 0;
    q.push_back( N * 2 + 1 );

    while ( !q.empty() ) {
        LL u = q.front();
        q.pop_front();
        for ( LL i = nexts[ u ]; i != -1; i = e[ i ].next ) {
            LL v = e[ i ].to;
            if ( e[ i ^ 1 ].capacity > 0 ) {
                if ( dist[ v ] > dist[ u ] - e[ i ].cost || dist[ v ] == -1 ) {
                    dist[ v ] = dist[ u ] - e[ i ].cost;
                    if ( !visited[ v ] ) {
                        if ( !q.empty() && dist[ q.front() ] > dist[ v ] )
                            q.push_front( v );
                        else
                            q.push_back( v );
                    }
                    visited[ v ] = true;
                }
            }
        }
        visited[ u ] = false;
    }
    return dist[ 0 ] != -1;
}

inline LL getmin_cost( LL u, LL op ) {
    visited[ u ] = true;
    if ( u == N * 2 + 1 ) return op;
    LL used, flow = 0;
    for ( LL i = nexts[ u ]; i != -1; i = e[ i ].next ) {
        if ( op <= 0 ) break;
        LL v = e[ i ].to;
        if ( !visited[ v ] && e[ i ].capacity > 0 &&
             dist[ v ] + e[ i ].cost == dist[ u ] &&
             ( used = getmin_cost( v, min( op, e[ i ].capacity ) ) ) > 0 ) {
            e[ i ].capacity -= used;
            e[ i ^ 1 ].capacity += used;
            op -= used;
            ans += used * e[ i ].cost;
            flow += used;
        }
    }
    return flow;
}

int main() {
    cin >> N;

    for ( int i = 1; i <= N; i++ ) {
        LL cap;
        cin >> cap;
        add( i, N * 2 + 1, cap, 0 ), add( 0, i + N, cap, 0 );
```

```
    }

    cin >> p >> m >> f >> n >> s;
    for ( int i = 1; i <= N; i++ ) {
        add( 0, i, inf, p );
        if ( N >= i + m ) add( i + N, i + m, inf, f );
        if ( N >= i + n ) add( i + N, i + n, inf, s );
        if ( N >= i + 1 ) add( i, i + 1, inf, 0 );
    }

    while ( spfa() ) {
        visited = vector<bool>( maxn, false );
        getmin_cost( 0, inf );
    }
    cout << ans << endl;
    return 0;
}

/*
Sample Input
3
1 7 5
11 2 2 3 1

Sample Output
134
*/
```

E : 节约粮食

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

Nasa 需要为 *m* 名 ACM 参赛选手准备食物，他决定为所有选手准备相同的套餐，套餐由 *n* 种不同的食物组成。由于口味不同，不同人吃一个单位的某种食物所获得的满足感可能不同，同时每个人的满足感是有上限的。为了不浪费食物，求在不超过任何人的满足感的前提下 Nasa 最多能花费多少钱。

Input

第一行包含 2 个正整数 *n*(3 ≤ *n* ≤ 20) 和 *m*(3 ≤ *m* ≤ 20)，代表食物的种类数和选手数

第二行包含 *n* 个实数，代表每种食物的单价

接下来的 *m* 行依次代表 *m* 名选手，每行包含 *n* + 1 个实数。每行的前 *n* 个数中，第 *i* 个数代表该名选手从一个单位的第 *i* 种食物中所获得的满足感。第 *n* + 1 个数为该名选手的满足感上限。

Output

输出 “Nasa can spend xxx taka.” （不带引号），其中 xxx 为一个**整数**（向上取整），代表 Nasa 的最大花费

Sample Input

```
3 3
1 0.67 1.67
1 2 1 430
3 0 2 460
1 4 0 420
```

Sample Output

```
Nasa can spend 1354 taka.
```

Hint

注意浮点误差

```
/*
E - 节约粮食
 */
#include <bits/stdc++.h>
using namespace std;

namespace simplex {

using Array = vector<double>;
using Matrix = vector<Array>;

const double eps = 1e-8, inf = (double) ( 1ll << 60 );
// 系数矩阵，第一行为c，第一列为b
Matrix A;

class Solver {
  private:
    int m;
    int n;

  public:
    Solver( int _m, int _n )
        : m( _m )
        , n( _n ) {}

    void input_matrix( const Matrix &a, const Array &b, const Array &c ) {
        for ( int i = 1; i <= n; ++i ) {
            A[ 0 ][ i ] = c[ i - 1 ];
        }
        for ( int i = 1; i <= m; ++i ) {
            for ( int j = 1; j <= n; ++j ) {
                A[ i ][ j ] = a[ i - 1 ][ j - 1 ];
            }
            A[ i ][ 0 ] = b[ i - 1 ]; // input b
        }
    }

    void display() const {
        cout << "m=" << m << ", n=" << n << endl;
```

```
    for ( int i = 0; i <= m; ++i ) {
        for ( int j = 0; j <= n; ++j ) {
            cout << "\t" << A[ i ][ j ];
        }
        cout << endl;
    }
}

void pivot( int l, int e ) {
    double tmp = A[ l ][ e ];
    A[ l ][ e ] = 1.0;
    vector<int> tmp_arr;
    for ( int j = 0; j <= n; ++j ) {
        if ( std::fabs( A[ l ][ j ] ) > 0 ) {
            A[ l ][ j ] /= tmp;
            tmp_arr.push_back( j );
        }
    }
    int cnt = tmp_arr.size();
    for ( int i = 0; i <= m; i++ ) {
        if ( i != l && std::fabs( A[ i ][ e ] ) > eps ) {
            tmp = A[ i ][ e ], A[ i ][ e ] = 0;
            for ( int j = 0; j < cnt; ++j ) {
                A[ i ][ tmp_arr[ j ] ] -= A[ l ][ tmp_arr[ j ] ] * tmp;
            }
        }
    }
}

bool simplex( bool verbose = false ) {
    while ( true ) {
        int l = 0, e = 0;
        double minv = inf;
        for ( int j = 1; j <= n; ++j ) {
            if ( A[ 0 ][ j ] > eps ) {
                e = j;
                break;
            }
        }
        if ( !e ) break;
        for ( int i = 1; i <= m; ++i ) {
            if ( A[ i ][ e ] > eps && A[ i ][ 0 ] / A[ i ][ e ] < minv ) {
                minv = A[ i ][ 0 ] / A[ i ][ e ];
                l = i;
            }
        }
        if ( !l ) {
            // printf( "Unbounded\n" );
            return false;
        }
        pivot( l, e );

        if ( verbose ) display();
    }
    return true;
}
```

```
    double get_result() const { return -A[ 0 ][ 0 ]; }

    void standardize() {
        for ( int i = 0; i < n; ++i )
            A[ 0 ].push_back( 0 );
        for ( int i = 1; i <= m; ++i ) {
            for ( int j = 1; j <= m; ++j ) {
                if ( i == j ) {
                    A[ i ].push_back( 1 );
                } else {
                    A[ i ].push_back( 0 );
                }
            }
        }
        n += m;
    }
};

} // namespace simplex

int main() {
    using namespace simplex;

    int n, m;
    cin >> n >> m;

    A = Matrix( m + 1, Array( n + 1 ) );
    for ( int i = 1; i <= n; ++i ) {
        cin >> A[ 0 ][ i ];
    }
    for ( int i = 1; i <= m; ++i ) {
        for ( int j = 1; j <= n; ++j )
            cin >> A[ i ][ j ];
        cin >> A[ i ][ 0 ];
    }

    Solver s( m, n );
    // s.display();

    s.simplex();
    // s.display();

    auto result = (long long) ( floor( s.get_result() * m + 1 - 0.0001 ) );
    cout << "Nasa can spend " << result << " taka." << endl;
    return 0;
}

/*
Sample Input
3 3
1 0.67 1.67
1 2 1 430
3 0 2 460
1 4 0 420

Sample Output
Nasa can spend 1354 taka.
*/
```

1050 网络流

A : 最大流问题

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

给出一张网络有向图，源点及汇点，计算其最大流。

Input

第一行给出四个整数N(1 <= N <= 200), M(N <= M <= 5000),S,T，分别表示节点数量、有向边数量、源点序号、汇点序号。

接下来M行每行包括三个正整数ui，vi，wi，表示第 i条有向边从 ui出发，到达 vi，边权为 wi。

Output

一行，包括一个正整数，为该网络流最大流。

Sample Input

```
6 9 4 2
1 3 10
2 1 20
2 3 20
4 3 10
4 5 30
5 2 20
4 6 20
5 6 10
6 2 30
```

Sample Output

```
50
```

```
/*
A - 最大流问题

reference:
https://oi-wiki.org/graph/flow/max-flow/
 */
#include <bits/stdc++.h>
using namespace std;

namespace dicnic {
const int maxn = 250;
const int INF = 0x3f3f3f3f;

struct Edge {
```

```
    int from, to, cap, flow;
    Edge( int u, int v, int c, int f )
        : from( u )
        , to( v )
        , cap( c )
        , flow( f ) {}
};

struct Dinic {
    // n：点数，m：边数，s：源，t：汇
    int n, m, s, t;
    // edges：所有边的集合
    vector<Edge> edges;
    // G：点 x -> x 的所有边在 edges 中的下标
    vector<int> G[ maxn ];
    int d[ maxn ], cur[ maxn ];
    bool vis[ maxn ];

    Dinic( int _n, int _m, int _s, int _t )
        : n( _n )
        , m( _m )
        , s( _s )
        , t( _t ) {
        init( n );
    }

    void init( int n ) {
        for ( int i = 0; i < n; i++ )
            G[ i ].clear();
        edges.clear();
    }

    void AddEdge( int from, int to, int cap ) {
        edges.push_back( Edge( from, to, cap, 0 ) );
        edges.push_back( Edge( to, from, 0, 0 ) );
        m = edges.size();
        G[ from ].push_back( m - 2 );
        G[ to ].push_back( m - 1 );
    }

    bool BFS() {
        memset( vis, false, sizeof( vis ) );
        queue<int> Q;
        Q.push( s );
        d[ s ] = 0;
        vis[ s ] = true;
        while ( !Q.empty() ) {
            int x = Q.front();
            Q.pop();
            for ( int i = 0; i < G[ x ].size(); i++ ) {
                Edge &e = edges[ G[ x ][ i ] ];
                if ( !vis[ e.to ] && e.cap > e.flow ) {
                    vis[ e.to ] = true;
                    d[ e.to ] = d[ x ] + 1;
                    Q.push( e.to );
                }
            }
        }
```

```
        return vis[ t ];
    }

    int DFS( int x, int a ) {
        if ( x == t || a == 0 ) return a;
        int flow = 0, f;
        for ( int &i = cur[ x ]; i < G[ x ].size(); i++ ) {
            Edge &e = edges[ G[ x ][ i ] ];
            if ( d[ x ] + 1 == d[ e.to ] &&
                 ( f = DFS( e.to, min( a, e.cap - e.flow ) ) ) > 0 ) {
                e.flow += f;
                edges[ G[ x ][ i ] ^ 1 ].flow -= f;
                flow += f;
                a -= f;
                if ( a == 0 ) break;
            }
        }
        return flow;
    }

    // s: 源, t: 汇
    int Maxflow( int s, int t ) {
        this->s = s;
        this->t = t;
        int flow = 0;
        while ( BFS() ) {
            memset( cur, 0, sizeof( cur ) );
            flow += DFS( s, INF );
        }
        return flow;
    }

    int Maxflow() { return Maxflow( s, t ); }
};
} // namespace dicnic

int main() {
    using namespace dicnic;

    int N, M, S, T;
    cin >> N >> M >> S >> T;

    Dinic dc( N, M, S, T );
    for ( int i = 0; i < M; ++i ) {
        int u, v, w;
        cin >> u >> v >> w;
        dc.AddEdge( u, v, w );
    }
    cout << dc.Maxflow() << endl;
    return 0;
}

/*
input:
6 9 4 2
1 3 10
2 1 20
2 3 20
```

```
4 3 10
4 5 30
5 2 20
4 6 20
5 6 10
6 2 30

output:
50
*/
```

B : 春秋战国运输线

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

公元前770年 - 公元前476年是春秋战国时期。

春秋时期，全国共分为一百四十多个大小诸侯国，而其中比较重要的有齐国、晋国、宋国、陈国、郑国、卫国、鲁国、曹国、楚国、秦国、吴国、越国、燕国等。

经过春秋时期（公元前770年—公元前476年，一说公元前453年，另一说公元前403年）的旷日持久的争霸战争，周王朝境内的诸侯国数量大大减少，公元前453年，韩、赵、魏推翻智氏，以三家分晋的结果为标志，奠定了战国七雄的格局。

到了战国中期，剩下来的七个主要大国秦、楚、韩、赵、魏、齐、燕被称为战国七雄。

齐,楚两个国家正在交战，其中楚国的物资运输网中有N个中转站，M条单向道路。

设其中第i条道路连接了ui，vi两个中转站，那么中转站ui可以通过该道路到达vi中转站，如果切断这条道路，需要代价wi。

现在齐国想找出一个路径切断方案，使中转站s不能到达中转站t，并且切断路径的代价之和最小。

Input

第一行给出四个整数N(1 <= N <= 200), M(N <= M <= 5000),S,T，分别表示中转站的数量、单向道路数量、中转站S、中转站T。

接下来M行每行包括三个正整数ui，vi，wi，表示第 i条单向道路从 ui出发，到达 vi，切断路径的代价为 wi。

Output

一行，包括一个正整数，为切断路径的代价之和最小。

Sample Input

```
6 7 1 6
1 2 3
1 3 2
2 4 4
2 5 1
3 5 5
4 6 2
5 6 3
```

Sample Output

```
5
```

```cpp
/*
B - 春秋战国运输线

reference:
最小割问题：https://oi-wiki.org/graph/flow/min-cut/
 */
#include <bits/stdc++.h>
using namespace std;

const int N = 1e4 + 5, M = 2e5 + 5;
int n, m, s, t, tot = 1;
int lnk[ N ], ter[ M ], nxt[ M ], val[ M ], dep[ N ], cur[ N ];

void add( int u, int v, int w ) {
    ter[ ++tot ] = v, nxt[ tot ] = lnk[ u ], lnk[ u ] = tot, val[ tot ] = w;
}

void addedge( int u, int v, int w ) { add( u, v, w ), add( v, u, 0 ); }

int bfs( int s, int t ) {
    memset( dep, 0, sizeof( dep ) );
    memcpy( cur, lnk, sizeof( lnk ) );
    queue<int> q;
    q.push( s ), dep[ s ] = 1;
    while ( !q.empty() ) {
        int u = q.front();
        q.pop();
        for ( int i = lnk[ u ]; i; i = nxt[ i ] ) {
            int v = ter[ i ];
            if ( val[ i ] && !dep[ v ] ) q.push( v ), dep[ v ] = dep[ u ] + 1;
        }
    }
    return dep[ t ];
}

int dfs( int u, int t, int flow ) {
    if ( u == t ) return flow;
    int ans = 0;
    for ( int &i = cur[ u ]; i && ans < flow; i = nxt[ i ] ) {
        int v = ter[ i ];
        if ( val[ i ] && dep[ v ] == dep[ u ] + 1 ) {
            int x = dfs( v, t, std::min( val[ i ], flow - ans ) );
```

```
            if ( x ) val[ i ] -= x, val[ i ^ 1 ] += x, ans += x;
        }
    }
    if ( ans < flow ) dep[ u ] = -1;
    return ans;
}

int dinic( int s, int t ) {
    int ans = 0;
    while ( bfs( s, t ) ) {
        int x;
        while ( ( x = dfs( s, t, 1 << 30 ) ) )
            ans += x;
    }
    return ans;
}

int main() {
    scanf( "%d%d%d%d", &n, &m, &s, &t );
    while ( m-- ) {
        int u, v, w;
        scanf( "%d%d%d", &u, &v, &w );
        addedge( u, v, w );
    }
    printf( "%d\n", dinic( s, t ) );
    return 0;
}

/*
input:
6 7 1 6
1 2 3
1 3 2
2 4 4
2 5 1
3 5 5
4 6 2
5 6 3

output:
5
*/
```

C : 最小割点

Time Limit: `1 Sec`, Memory Limit: `128 Mb`

Description

学过网络流我们知道，一个网络的最小割等于其最大流。

但这是针对最小割边而言。

我们看这么个问题：

两台电脑通过一系列诸如交换机、路由器等网络设备通信，网络可以有多种链路连通。这两台电脑只要还有网络设备连通它们，就能互相通信。

给定它们的相连关系，至少坏掉几个网络设备，会使这两台电脑无法通信?

Input

每组数据给出 n 和 m，表示 n 个设备，m个相连关系。

设备编号为 1, 2, ..., n。 其中 1、2 号设备为题述的两台电脑。

数据保证最初两台电脑连通。

$3 \le n \le 100$，$1 \le m \le 1000$。

Output

使两台电脑无法通信的最少坏掉的设备个数。

Sample Input

```
4 3
1 3
2 4
3 4
```

Sample Output

```
1
```

Hint

最小割点可以拆成最小割边问题处理：

对于本题，两个节点相连，比如 1 与 3 相连，相当于有一个 `1 -> 3` 的边，和一条 `3 -> 1` 的边。

拆点，可以给每个点两个编号，一个负责入边，一个负责出边，把 `1` 拆成 `1 * 2 = 2` 与 `1 * 2 + 1 = 3`，把 `3` 拆成 `3 * 2 = 6` 与 `3 * 2 + 1 = 7`

建新图：`2 -> 3, 6 -> 7, 3 -> 6, 7 -> 2`

对新图求最小割边，就是原图的最小割点了。

```cpp
/*
C - 最小割点
*/
#include <bits/stdc++.h>
using namespace std;

using LL = long long;
using Array = vector<int>;

const int maxn = 1e4 + 3, maxm = 1e5 + 3;
const LL inf = 1ll << 62;

int n, m, s, t, cnt;
Array head( maxn ), d( maxn ), cur( maxn );

struct Edge {
    int to, next;
```

```
    LL cap;
    Edge( int to = -1, int next = -1, int cap = -1 )
        : to( to )
        , next( next )
        , cap( cap ) {}
} edge[ maxm << 1 ];

void Link( int u, int v, int w ) {
    edge[ ++cnt ] = Edge( v, head[ u ], w );
    head[ u ] = cnt;
}

void addlink( int u, int v, int w ) { Link( u, v, w ), Link( v, u, 0 ); }

inline bool bfs() {
    queue<int> q;
    for ( int i = 0; i <= n; i++ )
        cur[ i ] = head[ i ];
    d = Array( maxn, 0 );
    q.push( s ), d[ s ] = 1;
    while ( q.size() ) {
        int u = q.front();
        q.pop();
        for ( int i = head[ u ]; i; i = edge[ i ].next ) {
            int v = edge[ i ].to, w = edge[ i ].cap;
            if ( d[ v ] || !w ) continue;
            d[ v ] = d[ u ] + 1;
            q.push( v );
        }
    }
    return d[ t ];
}

LL dfs( int u, LL flow ) {
    if ( u == t ) return flow;
    LL rest = flow, f;
    for ( int i = cur[ u ]; i && rest; i = edge[ i ].next ) {
        cur[ u ] = i;
        int v = edge[ i ].to;
        LL w = edge[ i ].cap;
        if ( d[ u ] + 1 != d[ v ] || w <= 0 ) continue;
        f = dfs( v, min( rest, w ) );
        if ( f == 0 ) d[ v ] = 0;
        rest -= f;
        edge[ i ].cap -= f;
        edge[ i ^ 1 ].cap += f;
    }
    return flow - rest;
}

LL Dinic() {
    LL max_flow = 0, f;
    while ( bfs() )
        while ( f = dfs( s, inf ) )
            max_flow += f;
    return max_flow;
}
```

```
int main() {
    while ( cin >> n >> m ) {
        head = Array( maxn, 0 );
        cnt = 1;
        n = 2 * n + 1;
        while ( m-- ) {
            int u, v;
            cin >> u >> v;
            addlink( 2 * u, 2 * u + 1, 1 );
            addlink( 2 * v, 2 * v + 1, 1 );
            addlink( 2 * u + 1, 2 * v, 1 );
            addlink( 2 * v + 1, 2 * u, 1 );
        }
        s = 2, t = 5;
        cout << Dinic() << endl;
    }
    return 0;
}

/*
Sample Input
4 3
1 3
2 4
3 4

Sample Output
1
*/
```