Ngày 1/12/2025: Quy hoạch động: BÀI TOÁN SUBSETSUM

Bài 3: Chọn tổng (SubsetSum)

Không thể dùng mảng 2 chiều như bài 1 được
⇒ Cách làm:

Bài 4: Rút gọn dãy số (Reduce)

Bài này khá khó ở phần truy vấn 🙂
SOLUTION:
Quy bài toán về việc đặt dấu +,- giữa các số
⇒ Có thể xếp n-1 dấu + và 1 dấu - không?
Có thể xếp n-2 dấu + và 2 dấu - không?
Có thể xếp n-3 dấu + và 3 dấu - không?
…..
Có thể xếp 1 dấu + và n-1 dấu - không?

⇒ Câu trả lời là có. Chỉ cần tồn tại cả 2 loại dấu trong dãy dấu đã cho (cả dấu + và dấu -). Không thể xếp n dấu cộng hoặc n dấu trừ
⇒ **Nếu dãy dấu như sau + 5 - 3 - 4 + 2 + 1 + 6 + 7 - 10**
Thì mình có thể viết lại thành + 5 - 3 - (4 - 2 - 1 - 6 - 7) - 10
Thì cách thu hoạch từ dãy dấu trên có thể như sau: 2 2 2 2 0 0 0

⇒ Miễn là tồn tại cả 2 loại dấu thì hoàn toàn có thể tạo ra dãy dấu +,- từ đó tạo ra được dãy kết quả đề yêu cầu

Vì đề bài đảm bảo có thể tạo ra được số V từ dãy đã cho. Và a[i] là số nguyên dương (max(a[i:1->n] <= 200). Đặt k = max(a[i:1→n])*n
⇒ -k < V < k

Vì nếu V < -k hoặc V > k thì sẽ không thể tạo ra số V. Tuy nhiên nếu V = -k hoặc V = k thì phải xếp dấu - hoặc dấu + vào tất cả các số **(Không thỏa mãn theo yêu cầu ở trên)**

Vì đã được đề bài đảm bảo nên mình gọi dp[i][S] = 0 hoặc 1 (= 0 nếu không thể tạo ra tổng S từ i phần tử đầu, = 1 nếu ngược lại)

Từ mảng dp, mình có thể tạo ra mảng d[i]. d[i] = -1 nếu mình xếp dấu - trước phần tử i, d[i] = 1 nếu mình xếp dấu + trước phần tử i (Vận dụng kiến thức của bài 1)

⇒ Từ mảng d[i], mình cần tạo ra 1 chuỗi các kết quả truy vấn sao cho hợp lý.

⇒ **Có thể Dùng deque để xử lý**
Gọi End là số hiêu phần tử kết thúc cuối cùng. Ví dụ như test đã cho ở trên, thì End = 0.
Hoặc với mảng d[i] sau: -1 -1 1 1 -1 1 Thì End = 3
⇒ Dãy trên sẽ truy vấn được kết quả 1 3 2 2 1
Hay nói Cách khác. Phần tử End là phần tử đầu tiên sao cho d[End]=1 và d[End]=-1

⇒ Cần xử lý từ 1→ End và từ End+1 → n (Các bạn sẽ có nhiều cách để xử lý) - Code mẫu và giải thích cách của mình sau đây

```
int FindEnd()
{
    for (int i = 0 ; i < n ; ++i) if(d[i]>0&&(i==n-1||d[i+1]<0)) return i;
}
```

```cpp
void Print()
{
    int End = FindEnd();
    deque<int> v;
    for (int i = 0 ; i < End ;i++)
    {
        v.pb(i);
        if(d[i]<0)
        // Chỉ xử lý trường hợp -1 -1 -1 1 1 1
        // Đảm bảo đoạn đầu sẽ có dạng 1 1 1 (1) -1
        // Hoặc dạng -1 -1 -1 (1) -1
        {
            int j = i+1;
            while(j<End&&d[j]>0)
            {
                cout << v.size()-1 << ' '  ;
                j++;
            }
            i = j-1;
        }
    }
    v.pb(End);
```

```cpp
if(End==n-1)
// End == n-1 <=> d: -1 -1 .... 1
{
    while(v.size()>1)
    {
        cout << v.size()-1 << ' ' ;
        // Cout vị trí của phần tử cuối sau mỗi lượt rút
        v.pop_front();
    }
    return;
}

   for (int i = End+1; i < n; )
   {
       v.pb(i);
       if(d[i]<0)
       {
           int j = i+1;
           while(j<n&&d[j]>0)
           {
               cout << v.size()-1 << ' ' ;
               // Ví dụ .... -1 1 1 1 -1
               // Có thể rút gọn ngay thành
               // .... -(1 -1 -1 -1) -1
               j++;
           }
           i = j;
       }
       if(i==n)
       {
           if(d[v.front()]==-1)
           {
               // Nếu mảng d có dạng -1 -1 -1 (1) ....
               // Với (1) là phần tử End
               v.pop_back();

               cout << v.size()-1 << ' ' ;
               while(v.size()>1)
               {
                   cout << v.size()-1 << ' ' ;
                   v.pop_front();
               }
               return;
           }
           else
           {
               // Nếu mảng d có dạng 1 1 1 (1) -1....
               // Với (1) là phần tử End
               while(v.front()!=End)
               {
                   cout << v.size()-1 << ' ' ;
                   v.pop_front();
               }
               cout << 0;
           }
           return;
       }
       v.pop_back();
       cout << v.size()-1 << ' ' ;
   }
```

Ngày 24/11/2025: Quy hoạch động: BÀI TOÁN LCS

Bài 4: Giá trị lớn nhất (NUMBER)

Nhận xét: Số lớn nhất, tức là số có độ dài dài nhất (Không tính số 0 ở đầu) và có thứ tự từ điển là lớn nhất
⇒ Vấn đề: Xử lý xâu chung có độ dài lớn nhất là phần cơ bản (Các bạn đã làm ở bài số 1) - Tuy nhiên ở bài này, vấn đề không nằm ở việc tìm xâu chung độ dài max, mà ở việc tìm truy vấn của nó.

Đầu tiên gọi lenMax[i][j] là độ dài **lớn nhất** của đoạn con chung từ a[i…n] và b[j…m] (n là độ dài của xâu a, và m là độ dài của xâu b)
⇒ Công thức:
lenMax[i][j] = max(lenMax[i+1][j],lenMax[i][j+1])
if(a[i]==b[j]) lenMax[i][j] = max(lenMax[i][j],lenMax[i+1][j+1]+1)
Công thức trên giống với bài 1 nên mình không bàn luận thêm

Gọi csLeng[L] là vector pair, lưu trữ chỉ số {i,j} trên 2 mảng a,b sao cho a[i]==b[j] và lenMax[i][j] = L

⇒ Sửa lại chút đoạn thiết lập lenMax ở trên để tích hợp việc add phần tử vào csLeng

```
lenMax[i][j] = max(lenMax[i+1][j],lenMax[i][j+1])
if(a[i]==b[j])
{
	lenMax[i][j] = max(lenMax[i][j],lenMax[i+1][j+1]+1)
	csLeng[lenMax[i][j]].pb({i,j});
}
```

Nhận xét: lenMax[0][0] chính là độ dài xâu con chung dài nhất giữa a và b ⇒ Tuy nhiên để chọn ra xâu thỏa mãn đề bài không dễ, vì phải xử lý các số 0 ở đầu (nếu có) và cần phải tìm số lớn nhất theo thứ tự từ điển.

Vậy mình sẽ lợi dụng mảng vector lenMax để truy vấn kiểu gì?
⇒ Với mỗi lenMax[L] ⇒ Sort lại theo cách tham lam của mình, phần tử nào đứng trước sẽ được ưu tiên cao hơn

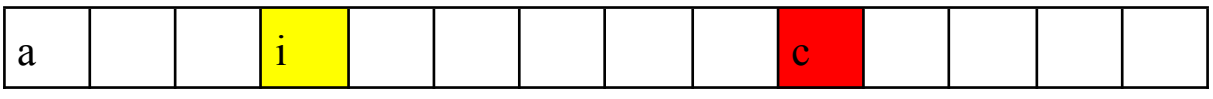
```
bool cmp ( const pair<int,int>&x, const pair<int,int>&y )
{
    if(a[x.first]==a[y.first]) return x.first+x.second < y.first+y.second;
    return a[x.first] > a[y.first];
}
```

⇒ Sau đó for ngược L từ lenMax[0][0] → 0. Lúc đầu, nếu chưa cout ra giá trị nào thì sẽ không được cout ra 0 (Phòng tránh trường hợp như ví dụ a = “1002” và b = “2001”)
⇒ Nếu tìm được giá trị đầu tiên thỏa mãn khác 0 thì sẽ cout ra, và các số đằng sau chọn phải đứng ở vị trí lớn hơn so với số đã được chọn

⇒ Tại sao cách tham lam như vậy lại đúng?

- Gọi {i,j} và {c,d} là 2 cặp chỉ số mình đang cần so sánh và lenMax[i][j] = lenMax[c][d] (Bởi vì cùng độ dài thì mình mới so sánh)
- Nếu a[i] > a[c] ⇒ Ưu tiên {i,j} hơn (Lớn hơn theo thứ tự từ điển)
- Nếu a[i]==a[c] ⇒ Xét i + j và c + d , nếu i + j < c + d thì ưu tiên i+j hơn

⇒ Tại sao?
Không mất tính tổng quát, giả sử {i,j} , {c,d} như hình vẽ sau



CHÚ Ý: a[i]==a[c]==b[d]==b[j]
⇒ lenMax[i][j] = lenMax[c][d]
⇒ **Vậy lenMax[i][d] sẽ như nào?**
Chắc chắn: lenMax[i][d] >= lenMax[i][j] và lenMax[c][d]
⇒ Tại sao? (Phần này các bạn tự ngẫm nhé)
Trường hợp 1: Nếu lenMax[i][d] == lenMax[i][j] == lenMax[c][d]
⇒ Lúc này thì cách tham lam mình hoàn toàn đúng, vì {i,d} sẽ cùng nằm trong vector với {i,j} và {c,d} mà a[i]==a[c]==b[d]==b[j]
⇒ Theo cách tham lam của mình thì {i,d} sẽ được ưu tiên hơn {c,d} và {i,j}

Trường hợp 2: Nếu lenMax[i][d] > lenMax[i][j] và lenMax[c][d]
⇒ i,d đã được xét ở độ dài lớn hơn nên mình không thể chọn i,d nữa (Các bạn ngẫm thêm ở đoạn này nhé)

CODE MẪU: Chú ý, việc SORT sẽ dẫn đến TLE ở test cuối. nên chúng ta phải tự code tay việc chọn cặp chỉ số tối ưu nhất

```cpp
pair<int,int> FindBestIJ ( int leng , pair<int,int> curEnd )
{
    char BestValue = char('0'-1);
    pair<int,int> cur;
    for ( auto pairIJ : csLeng[leng] )
    {
        if(curEnd.fi>=pairIJ.fi||curEnd.se>=pairIJ.se) continue;
        if(a[pairIJ.first]>BestValue)
        {
            BestValue = a[pairIJ.first];
            cur = pairIJ;
        }
        else if(a[pairIJ.first]==BestValue) cur = Min(cur,pairIJ);
    }
    return cur;
}

bool FindNumberStart ( int leng )
// return true nếu tìm được số đầu tiên thỏa mãn != 0
// return false nếu không thể tìm được phần tử != 0
{
    cur = {-1,-1};
    pair<int,int> x = FindBestIJ (leng,cur);
    if(a[x.first]=='0') return false;
    cur = x;
    cout << a[cur.first];
    return true;
}
```

```cpp
void Print()
{
    MaxLeng = lenMax[0][0];
    int flag = 0;
    while(MaxLeng>0) if(flag=FindNumberStart(MaxLeng--)) break;

    if(!flag) cout << 0;

    while(MaxLeng>0)
    {
        pair<int,int> x = FindBestIJ(MaxLeng,cur);
        // Tìm cặp {i,j} tối ưu nhất
        cur = x;
        cout << a[cur.fi];
        MaxLeng--;
    }
}
```

```cpp
void Do()
{
    cin >> a >> b;
    for ( int i = a.size()-1 ; i >= 0 ; --i )
        for ( int j = b.size()-1 ; j >= 0 ; j-- )
        {
            if(a[i]==b[j])
            {
                lenMax[i][j] = lenMax[i+1][j+1]+1;
                csLeng[lenMax[i][j]].pb({i,j});
            }
            lenMax[i][j] = max({lenMax[i][j],lenMax[i+1][j],lenMax[i][j+1]});
        }
    Print();
}
```

Bài 5: Dãy con chung (LCSK)

Để đọc solution bài này, các bạn nên đọc các kiến thức sau đây trước

- Priority queue (Hàng đợi ưu tiên) (Phần 1.3 trong bài này)
- Deque và kỹ thuật tìm min-max trong đoạn tịnh tiến

Trước khi đi đến thuật toán chuẩn, mình sẽ trình bày thuật toán trâu và từ đó dùng các kỹ thuật để cải tiến thuật toán trâu thành thuật chuẩn

Thuật trâu:

Định nghĩa mảng dp: Gọi dp[i][j] là Max độ dài xâu con chung thỏa mãn đề bài từ a[1→i] và b[1→j] **và 2 phần tử cuối là a[i] và b[j]**
⇒ Tức a[i] phải == b[j].
Nếu a[i] != b[j] ⇒ dp[i][j] = 0

→ For (i:1→n)
    For(j:1→m)
        if(a[i]!=b[j]) ⇒ dp[i][j] = 0
        else
            For(u:max(1,i-k) → i)
                For(v:max(1,j-k) → j)

dp[i][j] = max(dp[i][j],dp[u][v]+1)

⇒ Độ phức tạp o(n*m*k^2) ⇒ Rất lớn

<u>Thuật chuẩn</u>

Cải tiến việc chọn dp[u][v] Max sao cho u nằm trong khoảng [i-k,i] và v tương tự

Vì đang for 2 vòng lồng nhau (Trong trường hợp bỏ 2 vòng for k^2)
⇒ Tức là khi i tăng từ 1→n thì với mỗi i thì j đều tăng từ 1→m
Nghe khó hiểu phải ko?
Đơn giản chỉ là giả sử khi i = 1 ⇒ Lúc này j sẽ chạy từ 1,2,....,m
Sau đó i tăng lên 2 ⇒ Lúc này j lại chạy từ 1,2,...m
⇒ Ta giả sử: Khi j dịch chuyển thì i đang cố định

- Gọi pq[j] là priority queue (Hàng đợi ưu tiên) lưu trữ theo kiểu pair. Với phần tử first lưu giá trị của dp[u][j] và phần tử thứ hai = chỉ số u. (Với u nằm trong khoảng từ i-k tới vị trí i cố định đang xét)

⇒ pq[j].top().second là chỉ số u, sao cho dp[u][j] là lớn nhất trong tất cả u từ i-k → i

- Gọi dq là deque (Hàng đợi 2 đầu) lưu trữ theo kiểu pair. Với phần tử first lưu giá trị của dp[u][v] với u,v nằm trong khoảng i-k → i và j-k → j. Phần tử second lưu chỉ số v

Như vậy, trong tay đã đủ công cụ để xử lý bài này 🙂Vấn đề là chúng ta cần xoay sở nó như nào

```
For (i=1→n)
    For(j=1→m)
    {
        Nếu(a[i]==b[j])
        // Dùng kỹ thuật Min-Max để tìm ra thằng lớn nhất trong
        // deque ⇒ dp[i][j] = dq.front().first+1
```

// Lấy ra phần tử ở top của priority queue pq[j] (tất nhiên
// phải thỏa mãn điều kiện đề bài
// Sau đó add vào dq
// Và cuối cùng nếu a[i]==b[j] ⇒ push vào priority queue

// Lưu biến Max = max(Max,dp[i][j]) để lưu kết quả

}

Code mẫu: (độ phức tạp n^2 * log(n))

```
pair<int,int> FindBest ( int j )
{
    while( !dq.empty() && dq.front().second + k < j ) dq.pop_front();
    // Nếu dq.front.second (chỉ số của phần tử trong deque mà cách j
    // quá k thì pop_front luôn
    if(dq.empty()) return {0,0};
    return dq.front();
}

void Add ( pair<int,int> x )
{
    while( !dq.empty() && dq.back().first <= x.first ) dq.pop_back();
    // Vì x là thằng mới thêm vào nên x.second là chỉ số j là lớn nhất
    // Với những phần tử trong dq được thêm vào trước, thì chắc chắn
    // dq.back.second > x.second, mà nếu dq.back.first <= x.first thì
    // mình sẽ pop_back luôn, vì phần tử x sẽ tối ưu để mình tham lam hơn
    dq.push_back(x);
}
```

```cpp
pair<int,int> Chooseinpq ( int i , int j )
{
    while(true)
    {
        if(pq[j].empty())break;
        if(pq[j].top().se+k>=i) return pq[j].top();
        // top là dp[second][j] có giá trị lớn nhất
        // nếu chỉ số second trên mảng a, cách i không quá k đơn vị
        // thì mới chọn được
        pq[j].pop();
    }
    return {0,0};
}
```

```cpp
cin >> n >> m >> k ;
for (int i = 1 ; i <= n ; ++i) cin >> a[i];
for (int j = 1 ; j <= m ; ++j) cin >> b[j];

for (int i = 1 ; i <= n ; ++i,dq.clear())
    for ( int j = 1 ; j <= m ; ++j)
    {
        if(a[i]==b[j]) dp[i][j] = FindBest(j).fi + 1;

        auto x = Chooseinpq(i,j);
        if(x != make_pair(0,0)) Add({x.first,j});

        if(a[i]==b[j]) pq[j].push({dp[i][j],i});

        Max = max(Max,dp[i][j]);
    }

cout << Max ;
```

Ngày 17/11/2025: Giới thiệu về CT Truy hồi và quy hoạch động

Bài 3: Mô Hình (Boundary)

Nếu viên gạch i đặt theo chiều (a_i, b_i): chiều cao viên gạch = a_i; cạnh trên dài = b_i

Nếu đặt ngược (b_i, a_i): chiều cao = b_i ; cạnh trên dài = a_i

Để sang viên gạch bên cạnh (i → i+1):

- nó phải bò trên cạnh trên của viên thứ i
- sau đó bò 1 đoạn **đứng dọc theo mép tiếp xúc** để lên được cạnh trên của viên gạch i+1
  → chính là: $|cao[i+1]-cao[i]|$

⇒ Tổng đường đi gồm 3 phần:

- bò trên cạnh trên viên 1
- mỗi lần chuyển sang viên kế tiếp: $+|height[i]-height[i-1]|$

  → bò trên cạnh trên i

- bò trên cạnh trên viên n

→ Mục tiêu: chọn hướng xoay từng viên sao cho tổng độ dài đường đi lớn nhất

Đối với mỗi viên gạch i, ta có 2 trạng thái:

- fa[i] = độ dài lớn nhất nếu đặt cạnh đứng = a_i (tức chiều cao = a_i, cạnh trên = b_i)
- fb[i] = độ dài lớn nhất nếu đặt cạnh đứng = b_i (tức chiều cao = b_i, cạnh trên = a_i)

⇒ Nếu đang xét viên i:

Trường hợp 1: đặt theo chiều (a_i đứng, b_i ngang)

fa[i] = b_i + max(fa[i-1] + |b[i] - b[i-1]|, fb[i-1] + |b[i] - a[i-1]}|)

Trường hợp 2: đặt theo chiều (b_i đứng, a_i ngang)

fb[i] = a_i + max(fa[i-1] + |a[i] - b[i-1]|, fb[i-1] + |a[i] - a[i-1]}|)

⇒ Kết quả là max(fa[n] và fb[n])

Bài 4: Phân tích số (IPARD)

Từ bài toán gốc ban đầu, ta dễ dàng suy ra đề bài mới như sau: "Tính số cách chọn một tập hợp các số nguyên dương khác nhau sao cho tổng bằng n"
⇒ Gọi dp[n] = số cách để tạo tổng **n** bằng cách chọn các số phân biệt.
⇒ Sau đó, với mỗi số i từ 1 → n:

Ta cập nhật **ngược j từ n về i** (để tránh dùng một số nhiều lần)

Cụ thể:

```
for i = 1..n:
    for j = n..i:
        dp[j] += dp[j - i];
```

Tại sao lại phải như vậy? ⇒ Vì chúng ta for NGƯỢC i và j, Giả sử ta đang xét i (tức là với đồng tiền thứ i, thì các dp[j:1=>n] có thể tăng thêm bao nhiêu cách; hay nói cách khác là với việc dùng đồng tiền giá trị i, thì có bao cách để tạo ra các tổng j = 1…n)

⇒ Do vậy, nếu ta for theo chiều thuận thì giá trị i có thể dùng lặp lại nhiều lần (dp[x]+=dp[x-i]; tuy nhiên dp[x-i] đã bao gồm cách có chứa đồng tiền i ⇒ Nên xảy ra mâu thuẫn với đề bài ⇒ Cách này ko làm được

Bài 5: Trả tiền (COINS)

Bài này về ý tưởng dễ hơn so với bài 4 - Phân tích số (IPARD), tuy nhiên phần truy vết khá mới lạ và sẽ gây khó khăn cho các bạn

- Xử lý phần đếm số cách trước

Gọi dp[n] = số cách để tạo thành đúng số tiền n
⇒ Khởi tạo dp[0] = 1 ⇒ Cách để trả 0 đồng có 1 cách, đó chính là ko làm gì =))))

```
for i từ 1 đến 100:
    giá trị xu = i*i
    for j từ giá trị xu đến k:
        socach[j] += socach[j - giá trị xu]
```

⇒ Nhớ mod không sai nhé

- Tính số xu ít nhất cần trả và truy vết

Gọi soxumin[n] = số xu ít nhất để trả đúng n.

Khởi tạo:

- truyvet[n]: là xu nào sẽ được dùng để tạo lên n
- soxumin[0] = 0
- soxumin[s] = vô cùng lớn với mọi s > 0.

Cách làm:

Khi duyệt từng loại xu i^2, ta cập nhật:

```
nếu soxumin[j - i*i] + 1 < soxumin[j]:
    soxumin[j] = soxumin[j - i*i] + 1
    truyvet[j] = i    // tức là ta dùng 1 xu i² cho giá trị j
```

⇒ Sau khi có soxumin[k], ta phục hồi lại những đồng xu đã dùng:

```
tienthua = k
while tienthua > 0:
    i = truyvet[tienthua]
    giá trị xu = i*i
    demxu[i]++
    tienthua -= giá trị xu
```

→ Kết quả cuối cùng: demxu[i] = số lượng xu i^2 được dùng.

Cuối cùng in từ xu lớn → nhỏ theo chuẩn đề bài.

Ngày 10/11/2025: Chữa bài kiểm tra cuối khóa

Ngày 3/11/2025: Kiểm tra cuối khóa

Ngày 27/10/2025: Luyện tập

Bài 4: Đếm ước m*n

Để tìm số lượng ước của một số, ta cần **phân tích số đó ra thừa số nguyên tố**.

Giả sử:

m×n=p1^a1 × p2^a2 × ⋯ × pk^ak

Khi đó, **số lượng ước của m×n** được tính bằng:

(a1+1)×(a2+1)×⋯×(ak+1)

Bài 5: Số nguyên tố lớn nhất

Duyệt tất cả các khả năng cho hai dấu ?
Mỗi dấu ? có 10 khả năng (0–9) → tổng cộng 100 tổ hợp cần thử.
⇒ Thay các dấu ? trong chuỗi bằng hai chữ số cụ thể, tạo ra số đầy đủ. Với mỗi số sau khi thay xong thì kiểm tra xem số đó có phải là số nguyên tố không.
LƯU Ý: Giữ lại số nguyên tố lớn nhất trong tất cả các trường hợp thử.

Bài 6: Nguyên Tố

Duyệt toàn bộ chuỗi từ trái sang phải.
Tách liên tiếp các ký tự là chữ số (0–9) thành từng đoạn — đó là một số.
→ Khi gặp ký tự không phải số, ta kết thúc một đoạn số.
Kiểm tra xem số đó có phải là số nguyên tố không:
Có thể dùng sàng nguyên tố để đánh dấu trước các số nguyên tố

Nếu là số nguyên tố và chưa gặp trước đó, ta lưu lại vào danh sách.

Bài 7: Simple Math

<u>Thuật toán trâu:</u>
For 2 vòng (o(n^2)) ⇒ Với mỗi xâu con, lại tính độ độ đẹp của xâu con đó

<u>Thuật chuẩn:</u>
Nhận thấy, mình sẽ for theo độ dài.
Ví dụ: Để tính độ đẹp cho tất cả các xâu con của xâu có độ dài = 3, thì mình sẽ tính độ đẹp của

- Tổng số lượng kí tự nguyên âm của tất cả các xâu có độ dài = 1 rồi chia cho 1 (độ dài)
- Tổng số lượng kí tự nguyên âm của tất cả các xâu có độ dài = 2 rồi chia cho 2 (độ dài)
- Tổng số lượng kí tự nguyên âm của tất cả các xâu có độ dài = 3 rồi chia cho 3 (độ dài)
  ⇒ Cộng tổng 3 kết quả trên thì ra tổng độ đẹp của các xâu con có độ dài = 3

Ví dụ: xâu “YBI”
<u>Tính lần lượt theo quy ước trên</u>

- 2 / 1 (“Y”, “I”)
- 2 / 2 (“YB” , “BI”)
- 2 / 3 (“YBI”)
  ⇒ Tổng = 3.66667

Tính độ đẹp của tất cả các xâu có độ dài = 1 hoặc 2,.... n thế nào? Và làm thế nào chỉ cần làm trong o(n) ??

Gọi pre[i]: là số lượng kí tự nguyên âm từ đầu tới i
Nhận xét: (Theo hệ quy chiếu xâu bắt đầu từ 1 và kết thúc ở vị trí n)

- Tổng số lượng kí tự NÂ của xâu có độ dài 1 hoặc n = pre[n]-pre[0] ⇒ RES += (pre[n]-pre[0])/1 + (pre[n]-pre[0])/n
- Tổng số lượng kí tự NÂ của xâu có độ dài 2 hoặc n-1 = pre[n]-pre[0]+pre[n-1]-pre[1]
  ⇒ RES += (pre[n]-pre[0]+pre[n-1]-pre[1])/2 + (pre[n]-pre[0]+pre[n-1]-pre[1])/(n-1)
- Theo nguyên lý quy nạp: Làm tương tự với các độ dài lớn hơn, cho tới khi i > n/2

<u>GIẢI THÍCH tổng số lượng NÂ của xâu có độ dài 2 hoặc n-1</u>
Đặt mảng f với f[i] = 1 nếu s[i] là NÂ, = 0 nếu ngược lại
⇒ Tổng số lượng kí tự NÂ của xâu có độ dài = 2 là
(f[1]+f[2])+(f[2]+f[3])+....+(f[n-1]+f[n]) (Biểu thức 1)
⇒ Tổng số lượng kí tự NÂ của xâu có độ dài = n-1 là
(f[1]+f[2]+...+f[n-1]) + (f[2]+f[3]+.....+f[n]) (Biểu thức 2)

Ta nhận thấy:
Biểu thức 1 = Biểu thức 2
= (f[1]+f[2]+...+f[n-1]+f[n])+(f[2]+f[3]+...+f[n-1])
= pre[n]-pre[0]+pre[n-1]-pre[1]

CODE MẪU:

```
for(i=1; i * 2 <= s.length() + 1; i++){
   num+=s[n.length()-i+1]-s[i-1];
   // Dùng biến num để lưu trữ (Đỡ phải tính lại)
   res+=num/i;
   if((s.length()-i+1)!=i){
```

```
    res+=num/(s.length()-i+1);
   }
  }
```

NGày 13/10/2025: Mảng cộng dồn:

Bài 1: Tổng đoạn con:

Khái niệm đoạn con: Là một dãy các phần tử liên tiếp từ i -> j:

a1 a2 a3 a4 a5.

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N = 1e5 + 1;
int a[N], n, q, l, r;
int main()
{
    cin >> n;
    for(int i = 1; i <= n; i++)
        cin >> a[i];
    cin >> q;
    while(q--)
    {
        cin >> l >> r;
        long long sum = 0;
        for(int i = l ; i <= r; i++) sum += a[i];
        cout << sum << endl;
    }
}

```

Ta có thể cải tiến bằng cách chuẩn bị 1 mảng: **mảng cộng dồn:**

i	0	1	2	3	4	5	6	7			n
a		2	1	-3	4	2	0	2			
s	0	2	3	0	4	6	6	8			

Để tính tổng đoạn màu vàng S(3, 6):

S(3, 6) = s[6] - s[2]

S (2, 8) = s[8] - s[1]

S(l, r) = s[r] - s[l - 1]

Từ mảng a, ta tính mảng s:

s[0] = 0

s[1] = a[1] = s[0] + a[1]

s[2] = a[1] + a[2] = s[1] + a[2]

s[3] = a[1] + a[2] + a[3] = s[2] + a[3]

s[4] = a[1] + a[2] + a[3] + a[4] = s[3] + a[4]

s[i] = a[1] + a[2] +...+a[i] = s[i - 1] + a[i]

Lệnh để tính mảng s:

```
s[0] = 0;
for(int i = 1; i <= n; i++) s[i] = s[i - 1] + a[i];
```

Tương ứng với mỗi yêu cầu tính tổng từ l -> r:

s[r] - s[l - 1].

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N = 1e5 + 1;
int a[N], n, q, l, r;
long long s[N];
int main()
{
    cin >> n;
    s[0] = 0;
    for(int i = 1; i <= n; i++)
    {
        cin >> a[i];
        s[i] = s[i - 1] + a[i];
    }
    cin >> q;
    while(q--)
    {
        cin >> l >> r;
        long long sum = s[r] - s[l - 1];
        cout << sum << endl;
    }
}

```

Bài 2.

Cần tìm 1 đoạn con có tổng = m

Nếu có nhiều đoạn thỏa mãn, thì đưa ra đoạn có ít phần tử nhất.

i	0	1	2	3	4	5	6	7			n
a		2	1	-3	4	2	0	2			
s	0	2	3	0	4	6	6	8			

Có n đoạn con bắt đầu từ 1

Có n - 1 đoạn con bắt đầu từ 2.

...

có 1 đoạn con bắt đầu từ n.
Có bao nhiêu đoạn con tất cả?
n + (n - 1) + (n - 2) +...+ 1 = n(n+1)/2
Thuật đơn giản nhất:
Tính mảng cộng dồn s.

```
res = n + 1;
for(int i = 1-> n)
	for(int j = i -> n)
		///Tính tổng 1 đoạn từ i -> j:
		Sum = s[j] - s[i - 1];
		if(Sum == m)
			res = min(res, j - i + 1);
```

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll= long long;
const int N=1e5+1;
int a[N];
ll s[N], k;
int main()
{
    int m,n,Min=1e9+1;
    cin >> n >> m;
    s[0] = 0;
    for(int i=1; i<=n; i++)
    {
        cin>>a[i];
        s[i]=s[i-1]+a[i];
    }
    for(int i=1; i<=n; i++)
    {
       for(int j=i; j<=n; j++)
        {
            k = s[j] - s[i-1];
            if(k == m) Min = min(Min, j - i + 1);
        }
    }
    if(Min==1e9+1)cout<<-1;
    else cout<<Min;
}

```

(theo đề bài) Mảng a gồm các số nguyên dương mà s[i] = s[i - 1] + a[i]

-> Từ đó suy ra mảng s: s[i] > s[i - 1]

-> mảng s là mảng đã có thứ tự tăng dần.

i		1			j				
a		i	i	i				j	
s		j							

a[1] + a[2]+...+a[j] > m

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll= long long;
const int N=1e5+1;
int a[N];
ll s[N], k;
int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    int m,n,Min=1e9+1;
    cin >> n >> m;
    s[0] = 0;
    for(int i=1; i<=n; i++)
    {
        cin>>a[i];
        s[i]=s[i-1]+a[i];
    }
    int j = 1;
    for(int i=1; i<=n; i++)
    {
       while(j < n and s[j] - s[i - 1] < m) j++;
       if(s[j] - s[i - 1] == m)
           Min = min(Min, j - i + 1);

    }
    if(Min==1e9+1)cout<<-1;
    else cout<<Min;
}

```

Bài 3. Nấu ăn

Đề bài:

Danh mục số lượng các món ăn của mỗi người:

a1 a2 a3 theo như đề bài mô tả thì mỗi món ăn sẽ được đánh số báo danh theo quy định lần lượt

người 1: 1 -> a1
Người 2: a1 + 1 -> a1+a2
Người 3: a1 + a2 + 1 -> a1 + a2 + a3
Người 4: a1+a1+a3+1 -> a1 +a2+a3+a4
5 3
5 4 1 2 3
5
6
12

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

5 9 10 12 15

i	0	1	2	3	4	5
a		5	4	1	2	3
s	0	5	9	10	12	15

s chính là mảng cộng dồn của a: s[i] là món cuối cùng của người thứ i
Với mỗi giá trị p (món ăn trúng thưởng) ta cần xác định s[i] nhỏ nhất >= p
5 -> do người 1 nấu
6 -> do người 2 nấu
12: s[4] >= 12 -> do người 4 nấu
Phép tìm kiếm nhanh trên mảng tăng dần giá trị i nhỏ nhất sao cho s[i] >= p

int i = lower_bound(s, s + n + 1, p) - s;

re Untitled1.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll=long long;
const int N=1e5+1;
int a[N];
ll s[N],k;
int main()
{
    ios_base::sync_with_stdio(0);cin.tie(0);
    int m,n,p;
    cin>>n>>m;
    s[0]=0;
    for(int i=1; i<=n; i++)
    {
        cin>>a[i];
        s[i]=s[i-1]+a[i];
    }

    for(int i=1; i<=m; i++)
    {
        cin >> p;
        int k = lower_bound(s, s + n + 1, p) - s;
        cout << k << " ";
    }
}

```

Bài Tổng dãy số:

Cần xử lý đoạn biến a[i] thành số có 1 chữ số.
a[i] = i
a[123] = 123 -> Cần thuật toán tách chữ số để tính tổng các chữ số của a[i].
1 + 2 + 3 = 6.
a[123] = 6.
Phần còn lại sau khi đã xây dựng được mảng a thì nó là bài 1.

Bài Đếm giống:

Thay vì dùng 1 mảng cộng dồn ta dùng 3 mảng cộng dồn khác nhau tương ứng với 3 loại bò khác nhau
sau khi cộng dồn từng loại tương ứng
với mỗi câu hỏi l, r: ta in ra KQ của cả 3 mảng cộng dồn

Bài Sum1 = Tưới cây 1 (giống nhau)

Sử dụng mảng hiệu.
Gọi diff[i] = a[i]-a[i-1] (Chênh lệch giữa 2 thằng cạnh nhau)
Mỗi truy vấn l,r,x là tăng đoạn a[l],a[l+1],...,a[r] lên x

i	1	2	…	L	L+1	…	R	…	n
				+x	+x	+x	+x		

Nhận thấy là chênh lệch giữa 2 thằng cạnh nhau từ vị trí L ⇒ R là không thay đổi (tức diff[L+1],diff[L+2],...,diff[R] không thay đổi)
Tuy nhiên diff[L] và diff[R+1] thay đổi vì a[L] và a[R] tăng lên x, nhưng a[L-1] và a[R+1] không thay đổi
⇒ diff[L] tăng x đơn vị
⇒ diff[R+1] giảm x đơn vị

Cách code:
Thiết lập mảng diff ⇒ Khi cin mảng a, ta có mảng diff với diff[i] = a[i]-a[i-1]
Sau đó mỗi truy vấn l,r,x thì a[L] += x; a[R] -=x;
Sau q truy vấn, từ mảng diff, ta có thể tìm lại được mảng a.

- Vì diff[i] = a[i]-a[i-1] ⇒ diff[1] = a[1]-a[0] mà a[0] = 0 ⇒ diff[1] = a[1]
- Khi có a[i-1] ta hoàn toàn có thể tìm được thằng i với a[i] = diff[i]+a[i-1]

⇒ For(i=1;i<=n;++i) cout << a[i]

Kéo búa bao

Ban đầu ta tạo 3 mảng cộng dồn:
p[i] = số lần Cuội ra Bao (P) trong các ván 1 → i
s[i] = số lần Cuội ra Búa (S) trong các ván 1 → i
h[i] = số lần Cuội ra Kéo (H) trong các ván 1 → i
p[i] = p[i -1] + (a[i] == ‘P’);
s[i] = s[i - 1] + (a[i] == ‘S’);
h[i] = h[i - 1] + (a[i] == ‘H’)
Bây giờ, ta cần xét hết tất cả 6 trường hợp khi Bờm ra 1 loại từ 1→i, rồi đổi sang loại khác từ i+1→n:

Ví dụ: Khi xét đến điểm i bất kì, ta xét trường hợp:
Bờm ra H(Kéo) từ 1→i, đổi sang S(Búa) từ i+1→n:
Khi Bờm ra H, anh thắng khi Cuội ra P (Bao)
→ số trận thắng ở nửa đầu = p[i]

Khi Bờm ra S, anh thắng khi Cuội ra H (Kéo)
→ số trận thắng ở nửa sau = h[n] - h[i]
Tổng = p[i] + (h[n] - h[i])
Làm tương tự với 5 trường hợp còn lại(đổi từ búa → bao, búa sang kéo, …) và chọn ra trường hợp mà Bờm chơi thắng nhiều trận nhất
Lặp lại thao tác thử trên với mọi vị trí i từ 1 → n để ta bao quát được hết mọi trường hợp Bờm chơi từ 1→i rồi đổi lại chơi tiếp từ i + 1 → n

Dãy cân bằng

- Một cách xử lý bài này là ta một mảng mới b bằng cách
  + $b[i] = 1$ nếu $a[i]$ chẵn
  + $b[i] = -1$ nếu $a[i]$ lẻ
- Việc ta đang làm là thay số chẵn = 1 và thay số lẻ = -1 thì một dãy cân bằng trên dãy a sẽ có tổng bằng 0 trong dãy b
- Mà một dãy liên tiếp $[l, r]$ (từ l đến r) có tổng bằng 0 thì với sum là mảng cộng dồn của dãy b:
  $sum[r] - sum[l-1] = 0 <=> sum[r] = sum[l-1]$
- Mà ta muốn đếm số đoạn cân bằng chính là đếm số bộ hai chỉ số l, r mà có cùng giá trị trên mảng cộng dồn

Đoạn con có tổng lớn nhất

- Ta tạo mảng cộng dồn sum trên mảng a thì rõ ràng tổng các phần từ liên tiếp từ l đến r trong mảng a sẽ sử dụng công thức $sum[r] - sum[l-1]$. Giả sử ta cố định vị trí kết thúc r ta cần tìm đoạn con có tổng lớn nhất kết thúc tại r thì ta cần tham lam tìm $sum[l-1]$ có giá trị bé nhất thì $sum[r] - sum[l-1]$ có giá trị lớn nhất.
- Và thật ra ta có thể duy trì giá trị nhỏ nhất này bằng quan sat sau giả sử từ r chuyển sang $r+1$ thì giá trị $min(sum[l-1])$ với $l <= r$ nhỏ nhất sẽ thay đổi như nào ?
- Nó sẽ có thêm giá trị $sum[r]$ như sau:
  + Tại r thì $min(sum[l-1]) = min(sum[0], sum[1], ..., sum[r-1])$

+ Tại $r + 1$ thì
  $min(sum[l - 1]) = min(sum[0], sum[1], ..., sum[r - 1], sum[r])$

- Bằng cách cố định r và duy trì giá trị $min(sum[l - 1])$ với mỗi r, ta có thể tính đoạn con có tổng lớn nhất kết thúc tại r. Vậy với n giá trị r có thể ta lấy max mọi giá có thể thì thu được kết quả với độ phức tạp $O(n)$

Ảnh đẹp

- Ta thấy bài thực chất yêu cầu tìm dãy có tổng lớn nhất thỏa mãn có độ dài tối thiểu là 4 và độ dài chẵn (số cây mận = số cây đào)
- Thì thật ra nó giống bài tìm dãy có tổng lớn nhất trên nhưng ta chưa cập nhật $sum[i]$ vào biến lưu giá trị $sum[l - 1]$ nhỏ nhất vội mà ta để nó cập nhật sau 4 bước để đảm bảo dãy có tối thiểu 4 và chia thành hai loại biến là chẵn và lẻ.
- Các bạn có thể thấy ta có 2 biến min_chan, min_le để lưu giá trị nhỏ nhất ứng với chỉ số chẵn và lẻ (rõ ràng ta thấy để đoạn chẵn thì $l - 1$ và r phải cùng tính chẵn lẻ
- Và nhớ cập nhật bằng $sum[i - 4]$ thay vì $sum[i]$ giúp đảm bảo độ dài tối thiểu là 4

Đoạn giá trị lớn nhất

- Vẫn giống với ý tưởng của bài trước, chỉ khác là lần này ta sẽ cập nhật sau k bước thôi, tính trước f là mảng min dồn rồi với mỗi i ta sẽ tính res = prefix[i] - f[i - k] và lấy max của tất cả các res với mọi i

Đội hình thi đấu

- Ta nhận thấy vì chỉ cần 1 đội trưởng và $k - 1$ đội viên nên nếu chọn trước được đội trưởng ta sẽ tham lam chọn $k - 1$ đội viên còn lại có độ dẻo dai nhỏ nhất có thể để tiềm năng của đội là thấp nhất
- Ta sẽ sắp xếp n thành viên theo độ dẻo dai tăng dần thì nếu đội trưởng thuộc khoảng $[k, n]$ thì đương nhiên ta chọn $k - 1$ đội viên đầu vì sau khi sắp xếp lại thì họ là các đội viên ít dẻo dai nhất → Có thể sử dụng prefix sum để tính trước tổng độ dẻo dai của k - 1 thành viên đầu
- Vậy nếu đội trưởng nằm trong khoảng [k, n] thì ta sẽ chọn ai ? Đương nhiên là ta sẽ chọn người có sức mạnh nhỏ nhất trong khoảng [k, n] để tối thiểu hóa tiềm năng của đội → Dùng ý tưởng tương tự prefix sum để lưu trước người đó
- Còn nếu đội trưởng thuộc $k - 1$ đội viên đầu thì rõ ràng $k - 1$ đội viên dẻo dai nhất chính là từ $[1, k]$ và bỏ đi người đội trưởng hiện tại

- Trong trường hợp đội trường thuộc k - 1 người đầu, giả sử người ta chọn là i, nếu so sánh với cách chọn ở trên (chọn đội trưởng trong khoảng [k, n] thì ta sẽ lỗ a[i] - b[i] tiềm năng tổng của cả đội), vậy để tối thiểu hóa chi phí, ta sẽ chọn người có hiệu a[i] - b[i] nhỏ nhất. → Làm tương tự như trên, lưu trước người đó

→ Với mỗi k, ta phải thử cả 2 cách chọn đội trưởng ở trên và chọn ra cách tối ưu nhất

- Vậy ta tính được sức mạnh của cả đội với n người đội trưởng có thể thì độ phức tạp là $O(n)$

Biến đổi dãy số (CSEQ)

- Bài này là phiên bản khác của đoạn giá trị lớn nhất
- Ta nhận thấy khi ta thực hiện thao tác đổi dấu đoạn $[l, r]$ thì dấu hay giá trị các khoảng bên ngoài không đổi hay $[1, l-1]$ và $[r+1, n]$ không đổi. Lúc này nếu ta biểu diễn dưới dạng mảng cộng dồn sum thì nếu đổi dấu đoạn $[l, r]$ thì tổng n phần từ sẽ thành
  $sum[l-1] + (sum[n] - sum[r]) - (sum[r] - sum[l-1])$ lần lượt là
  + Tổng $l-1$ phần từ đầu
  + Tổng $n-r$ phần từ cuối
  + Tổng $r-l+1$ phần tử đổi dấu
- Thu gọn lại ta thu được
  $sum[n] - 2 * sum[r] + 2 * sum[l-1] = sum[n] - 2 * (sum[r] - sum[l-1])$
- Để tổng lớn nhất thì ta cần tìm tổng đoạn con liên tiếp $[l, r]$ nhỏ nhất có thể vậy ta chỉ cần làm ngược lại bài đoạn giá trị lớn nhất là được

Hình chữ nhật lớn nhất

- Ta sẽ quy bài này từ 2 chiều thành 1 chiều như sau
- Để xác định được 1 hình chữ nhật trước hết ta cố định hai hàng l, r tức là ta sẽ xét hết hình chữ nhật bắt đầu từ hàng l và kết thúc ở hàng r
- Việc này sẽ tốn $O(n^2)$ với mọi cặp l, r và ta còn phải xác định số cột nhưng lúc này khi đã xác định được hàng bắt đầu và kết thúc rồi thì với mỗi cột j ta sẽ lấy hết các phần tử thuộc hàng $[l, r]$ cho nên ta cần tạo mảng b với ý nghĩa $b_j = \sum a_{i,j}$ với $l <= i <= r$
- Vậy giá trị hình chữ nhật từ cột x đến cột y, hàng l đến hàng r chính là tổng của $b_l + b_{l+1} + \ldots + b_r$. Hay ta cần tính đoạn con lớn nhất trên mảng b việc

này tương tự bài trên ta có thể hoàn thành trong $O(n)$ vậy tổng độ phức tạp là $O(n^3)$

- Một **lưu ý** nhỏ là các bạn cần tính tổng phần từ thuộc các hàng $[l, r]$ cho mỗi cột (hay chính là các giá trị của mảng b) một cách nhanh chóng thì ta dễ dàng làm được bằng cách lưu một mảng cộng dồn theo mỗi cột tức là lưu n mảng cộng dồn cho n cột

Chia đất

Chọn một điểm (i,j) làm đỉnh chung của hai hình chữ nhật.
Có hai kiểu chia quanh đỉnh (i,j):
- Cách 1: một hình nằm trên-trái của (i,j), hình kia nằm dưới-phải của (i+1,j+1).
- Cách 2: một hình nằm trên-phải của (i,j), hình kia nằm dưới-trái của (i+1,j-1).

Với mỗi kiểu, ta cần đếm số đôi sao cho tổng hai hình chung đỉnh (i,j) bằng nhau.
Để nhanh, ta cần dùng prefix sum 2D để tính tổng
Giả sử ta duyệt theo cách 1 (cách 2 các bạn làm tương tự)
Bước 1: Duyệt tất cả hình chữ nhật ở phía trên trái lưu số lần xuất hiện của mỗi tổng vào cnt[sum].
Bước 2: Duyệt các hình chữ nhật ở phía dưới phải. Giả sử hình chữ nhật ở dưới phải có tổng = S, mà ta đã biết được số hình chữ nhất ở phía trên trái có tổng = S là cnt[S] (làm ở bước 1) ==> Lúc này chỉ cần tăng kết quả thêm cnt[S]

Lặp qua tất cả (i,j) và cả hai kiểu ⇒ tổng cuối cùng là kết quả.

Ví dụ cách code theo Cách 1 (Trên trái - Dưới phải)

```
for (i = 1 ; i <= n-1; ++i)
    for (j = 1 ; j <= n-1 ; ++j)
        //TẠO bảng đếm COUNT (rỗng)
        for (int x_up = 1 ; x_up <= i ; ++x_up)
            for( y_up = 1 ; y_up <= j; ++y_up)
            {
                S_up = GET_SUM(x_up, y_up, i, j) // Các bạn tự code hàm này
                COUNT[S_up] += 1
             }

        for ( int x_low = i+1 ; x_low <= n; ++x_low)
            for (int y_low = j+1 ; y_low <= n ; ++y_low)
            {
                S_low = GET_SUM(i+1, j+1, x_low, y_low)
                res += COUNT[S_low]   // có COUNT[S_low] cách chia hợp lệ
             }
```

->Độ phức tạp: Với n ≤ 50, mỗi (i,j) xét tối đa $O(n^2)$ hình chữ nhật 2 lần ⇒ tổng khoảng O(n^4) (vẫn chạy tốt với n nhỏ)

6/10/2025

Bài Nhà cao tầng

Duyệt từng hàng i (từ 1 đến m), tính rowMax[i] = giá trị lớn nhất trong hàng i.
Duyệt từng cột j (từ 1 đến n), tính colMax[j] = giá trị lớn nhất trong cột j.
Với mỗi ô (i,j), kiểm tra: nếu rowMax[i] > h[i][j] **và** colMax[j] > h[i][j] thì ô này **không hợp phong thủy** → cộng 1 vào đáp án.
In ra tổng các ô không hợp.

Bài Chọn Sách (BOOKS)

Ý tưởng chính: Ta sẽ **duyệt từ trái qua phải** để xác định Tí có thể lấy bao nhiêu quyển, và đồng thời tìm xem Tèo có thể lấy bao nhiêu quyển từ phải để tối ưu tổng cộng.

(Hiểu đơn giản, gọi Lmax là số sách lấy được tối đa của tí từ trái sang, thì ta sẽ thử xem nếu Tí lấy Lmax-1 thì Tèo lấy được bao nhiêu sách, nếu Lmax-2,...,0 ⇒ Tèo lấy được bao nhiêu sách)

Cách làm

Bước 1.

Duyệt từ **đầu dãy**, lấy lần lượt từng quyển cho Tí:

Nếu quyển tiếp theo có mã trùng với quyển đã chọn → dừng.
Nếu cộng giá vào mà vượt quá m → dừng.
→ Lúc này ta được số lượng sách ban đầu mà Tí có thể lấy, gọi là L.
Bước 2.

Duyệt tương tự từ **cuối dãy**, cho Tèo lấy từng quyển:

Nếu trùng mã với truyện của Tí → dừng.
Nếu tổng tiền vượt m → dừng.
→ Gọi số lượng sách Tèo lấy là R.

Bước 3.

Sau khi Tí và Tèo đã chọn, ta có thể **điều chỉnh lại lựa chọn của Tí**:

Tí bỏ dần 1 quyển từ bên trái,
Mỗi lần bỏ, thử để Tèo lấy thêm được nhiều quyển hơn ở cuối.
Tính lại tổng truyện = (sách Tí còn lại) + (sách Tèo lấy được)
→ Cập nhật kết quả lớn nhất.

Bước 4.

In ra kết quả cuối cùng.

Bài Ô chữ (PUZZLE)

Ý tưởng

1. **Tách từ theo hàng:**
   Duyệt từng hàng của bảng, ghép các chữ cái liên tiếp nhau lại thành từ.
   Khi gặp ký tự #, ta xem như kết thúc một từ.
   So sánh từ vừa thu được với từ lớn nhất đang có.
2. **Tách từ theo cột:**Duyệt từng cột của bảng, làm tương tự như khi duyệt hàng.
   Với mỗi chuỗi chữ liên tiếp (ngăn cách bởi #), ta cập nhật kết quả.
3. **Các bạn nên duy trì thêm biến res để lưu trữ string có thứ tự từ điển lớn nhất:** Nếu xâu mới tìm được > res thì gán res = xâu mới đó

4. **Kết quả cuối cùng:** Sau khi duyệt hết tất cả hàng và cột, từ lớn nhất thu được chính là kết quả cần in ra

Độ phức tạp: Duyệt mỗi ô tối đa 2 lần (1 lần theo hàng, 1 lần theo cột).
→ **Độ phức tạp O(m × n)**

29/9/2025

Vì các bạn đã AC hết bài 6, nên mình sẽ bắt đầu viết từ bài 7 nhé

Bài 7 (khôi phục ngoặc)

- Nhận xét:
- Vì đây là dãy ngoặc đúng → Với n dấu ngoặc đơn mở, mình cần phải bổ sung thêm n dầu ngoặc đóng → tổng 2*n dấu ngoặc
- Giả sử, ta biết dấu mở ngoặc ở vị trí i, có v dấu ngoặc ở giữa dấu ngoặc đó và dấu ngoặc đóng tương ứng, vậy chắc chắn vị trí thứ i+v+1 sẽ là vị trí của dấu ngoặc đóng tương ứng
- Cách code:
- Khởi tạo string kết quả, có 2*n phần tử
- For lần lượt giá trị trong dãy lúc đầu của đề bài, Dùng thêm biến Min_New (Vị trí nhỏ nhất mà vẫn chưa được điền)
- Lúc đầu Min_New = 0 ⇒ Sau đó điền dấu đóng,mở ngoặc đơn ⇒ Cập nhật lại Min_New….

Bài xóa chữ số

Bài tìm hoán vị

Bài biến đổi xâu

Cả 3 bài trên thầy đã chữa nên các bạn có thể xem lại ở phần slide được gửi trong zalo

22/9/2025

Bài 1

- <= (a+b-c)/3 team
- <= a/2 team
- <= b team

min ((a+b-c)/3, a/2, b)

Bài 2

Sắp xếp các số lại: a[1] <= a[2] <= a[3] <= … <= a[n]
Cách chọn tối ưu là lấy các phần tử ở đầu (bé nhất) => Lấy được bao nhiêu thì đáp án là bấy nhiêu

Trong TH không thể chọn được phần tử nào:

- k < 0 => Đáp án là -1
- k >= 0 => Đáp án là 0

5 -2
1 2 3 4 5

Bài 3

Ở bài này ta có thể dùng quay lui để sinh tất cả các số siêu nguyên tố trong khoảng từ a đến b

```cpp
bool check(long long x) //hàm kiểm tra số nguyên tố
{
    for(long long i = 2; i * i <= x; i++)
    {
        if(x % i == 0) return false;
    }
    return x >= 2;
}

void sinh(long long x)
{
    if(x > b || check(x) == false) return; //nếu số được sinh ra vượt quá b hoặc không phải là số nguyên tố thì quay lại
    if(x >= a && x > 10) ans++; //nếu số thỏa mãn và nằm trong khoảng [a, b]
    x *= 10; //tiếp tục sinh số mới bằng cách nhân với 10
    for(int i = 1; i <= 9; i += 2)
    {
        x += i; //thêm chữ số từ 1 đến 9 vào cuối số
        sinh(x); //gọi đệ quy để sinh tiếp
        x -= i; //trả lại giá trị ban đầu để tiếp tục vòng lặp
    }
    x /= 10; //trả về giá trị ban đầu trước khi thêm chữ số mới
}

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    cin >> a >> b;
    sinh(2ll); // bắt đầu sinh số nguyên tố từ 2
    sinh(3ll); // bắt đầu sinh số nguyên tố từ 3
    sinh(5ll); // bắt đầu sinh số nguyên tố từ 5
    sinh(7ll); // bắt đầu sinh số nguyên tố từ 7
    cout << ans;
}
```

Bạn nào chưa biết về thuật toán quay lui thì có thể tham khảo tại đây https://wiki.vnoi.info/algo/basic/backtracking.md

Bài 4

Đáp án là sqrt(r) - sqrt(l-1)

- [sqrt(x)] là số lượng số chính phương <= x

Bài 5

Tìm vị trí của 2 số giống nhau mà xa nhau nhất => Có nhiều cách

Cách 1: Sử dụng mảng đánh dấu

- first[v] = vị trí đầu tiên của v
- last[v] = vị trí cuối cùng của v

VD: 1 2 2 3 1 3 2

- first[1] = 1, last[1] = 5
- first[2] = 2, last[2] = 7
- first[3] = 4, last[3] = 6

=> max(last[v] - first[v] + 1)

Lưu ý: |Ai| <= 10^3 => -10^3 <= Ai <= 10^3

=> Cộng thêm 1000 vào mảng đánh dấu để chỉ số không âm.

Cách 2: Sắp xếp các số lại

- Các phần tử bằng nhau sẽ nằm cạnh nhau

Lưu chỉ số của số đó cùng với giá trị

VD: 1 2 2 3 1 3 2

[(1, 1), (1, 5), (2, 2), (2, 3), (2,7), (3,4), (3,6)]

Dãy bị chia làm nhiều vùng => Mỗi vùng chọn ra 2 phần tử xa nhau nhất

Xử lý:

```
for (int i = 1, j = 1; i <= n; ++i) {
        while (a[i] == a[j]) ++j;
        i = j - 1;
}
```

Bài 6

Sliding window

Gọi sum, cntEven là tổng của dãy đang xét và số lượng phần tử chẵn trong dãy đó

k phần tử

Khi chuyển từ dãy [i-k+1, i] -> [i-k, i+1]:

- Loại bỏ a[i-k+1]
- Thêm a[i+1]

Cách làm:

- Tính sum, cntEven cho a[1,2,...,k]
- a[1,..,k] -> a[2, …, k+1]
- a[2, …, k+1] -> a[3, …, k+2]
- a[3,...,k+2] -> a[4, …, k+3]
- …
- a[n-k, …, n-1] -> a[n-k+1,...,n]

=> Duyệt qua được tất cả các đoạn, tìm được đoạn có tổng lớn nhất

Bài 7

Nhận xét: Với giá trị S thỏa mãn → mọi S' > S đều thỏa mãn
Từ đây ta có thể nghĩ đến thuật toán chặt nhị phân
để tìm giá trị S nhỏ nhất khả thi
Câu hỏi đặt ra: Với mỗi S, làm sao để ta check xem là nó có khả thi hay không
→ Sử dụng thuật toán tham lam để check
Ta biết camera bắt buộc phải đặt vào 1 nhà nào đó, mà mỗi camera bao phủ được S khoảng cách về bên trái và S khoảng cách về bên phải

- Bắt đầu từ nhà xa nhất về bên trái chưa được phủ, gọi vị trí này là start

- Ta tìm nhà xa nhất so với nó, gọi là pos, sao cho a[pos] - a[start] <= S (vẫn trong phạm vi phủ của camera) rồi đặt camera vào vị trí pos
- Khi đó các nhà ở bên phải pos cách pos 1 khoảng <= S vẫn được bao phủ nên ta sẽ bỏ qua chúng
- Lặp lại quy trình này với 1 nhà khác chưa được phủ tới cho đến khi hết nhà hoặc hết camera

```
ll n, m, a[N];

bool check(ll mid)
{
    ll i = 1, cnt = 0;
    while(i <= n)
    {
        cnt++;
        if(cnt > m) return false;
        ll k = a[i] + mid;
        while(i <= n && a[i] <= k) i++;
        i--;
        k = a[i] + mid;
        while(i <= n && a[i] <= k) i++;
    }
    return true;
}

void ReadInput()
{
    cin >> n >> m;
    for(int i = 1; i <= n; i++) cin >> a[i];
}

void Solve()
{
    ll low = 1, high = 1e18, mid;
    while(low <= high)
    {
        mid = (low + high) / 2;
        if(check(mid)) high = mid - 1;
        else low = mid + 1;
    }
    cout << low;
}
```

Tìm kí tự

Ta để ý bài này giới hạn tuy nhỏ, nhưng khi cộng nhiều lần thì string f[i] của ta có thể trở nên rất dài, vượt quá giới hạn lưu trữ của c++ nếu làm theo cách thông thường
→ Thay vì ta tìm hẳn string f[n] thì ta sẽ tìm độ dài của string f[n] rồi từ độ dài ta tìm ra phần tử đứng ở vị trí thứ k
Vậy khi ta biết độ dài của xâu f[n] thì ta tìm phần tử thứ k kiểu gì ? Ở đoạn này ta sẽ sử dụng đệ quy để truy ngược về phần tử đó
Viết hàm FindChar(n, k) (với k tính từ 1):
Nếu n = 0 → trả về 'a'.
Nếu n = 1 → trả về 'b'.
Nếu n = 2 → trả về 'c'.
Nếu n ≥ 3:

Nếu k ≤ s[n-1] → ký tự nằm trong f(n-1).
→ Gọi FindChar(n-1, k).
Ngược lại nếu k ≤ s[n-1] + s[n-2] → ký tự nằm trong f(n-2).
→ Gọi FindChar(n-2, k - s[n-1]).
Ngược lại → ký tự nằm trong f(n-3).
→ Gọi FindChar(n-3, k - s[n-1] - s[n-2]).

Số dư (remainder)

⇒ cần tính xx…..x % m với n số x gần nhau
đặt v = 10^log10(x)
⇒ xx…..x = (x*v^n+x*v^(n-1)+....+x*v+x*v^0)%m
= x*(v^n+v^(n-1)+...+x*v^0)%m
= (x%m)*(v^n+v^(n-1)+...+x*v^0)%m

Đặt SD = (v^n+v^(n-1)+...+x*v^0)%m
⇒ Kết quả của bài = (x%m)*SD
⇒ Dễ thấy rằng: ta có thể tính ra SD bằng đệ quy
Trong trường hợp

1. n là số lẻ ⇒ v^n + v^(n-1) + …. + v^(n/2+1) + v^(n/2) + ..+ v^0

⇒

v^(n/2+1)*(v^(n/2)+..+v^0)+..+v^(n/2)+...+v^0

⇒ (v^(n/2+1)+1)*(v^(n/2)+....+v^0)

2. n là số chẵn

⇒ v^n + v^(n-1)+....+v^0 = v^n + (v^(n-1)+...v^0) ⇒ Đẩy về trường hợp số lẻ

Cho táo

Gọi End là **số táo cuối cùng** mà *mỗi* người sẽ có sau khi kết thúc quá trình (tức là mọi người đều có End quả táo).
⇒ Nếu ta biết End, ta có thể **làm ngược lại** để tính ra **số táo ban đầu** của mỗi người
Cách làm ngược lại: bắt đầu với a[i] = End cho mọi i. Lần lượt từ i = n về 1, ta "bỏ ngược" hành động của người i trong bước thuận

⇒ Trong bước thuận, người i đã cho mỗi người khác một lượng bằng số táo của người đó tại thời điểm cho. Ngược lại, khi đi ngược, mỗi người khác j sẽ trả lại a[j]/2, nên ta cần a[j] hiện tại phải chẵn; ta cộng tổng sum các a[j]/2 vào a[i] và đặt a[j] = a[j]/2.
Nếu có a[j] lẻ ở bất kỳ bước nào → T không khả dĩ (không

cho ra nghiệm nguyên).
Sau khi đi ngược lại hết, ta thu được a[1] (số táo ban đầu của người 1) — gọi là A1(T) nếu tồn tại.

Nhận xét rằng: khi xem A1(End) như hàm của End thì nó là hàm tuyến tính tỉ lệ (các phép chia và cộng đều tuyến tính theo End). Do đó A1(End) là đơn điệu không giảm theo End. (Nói ngắn: nếu tăng End, các a[i] cuối cùng tăng tỉ lệ, dẫn đến A1(End) không giảm.)

Hay nói cách khác: Nếu End tăng thì số táo của anh Quân lúc đầu sẽ tăng, Nếu End giảm thì số táo của anh Quân lúc đầu sẽ giảm.

=> Ta có thể dùng **binary search trên biến End** để tìm End sao cho A1(End) == m (m là số táo ban đầu của Quân cho trước).
Nếu không tồn tại End nguyên thỏa mãn A1(End) == m → in -1.

15/9/2025

Cặp giá trị khác nhau

Gọi cnt[x] là số lần xuất hiện của số x.
Ta cần đếm số cặp (i,j) sao cho a[i]-a[j]=k. Nghĩa là với mỗi phần tử a[j], ta cần tìm xem có bao nhiêu a[i] = a[j]+k

Công thức:

$$\text{ans} = \sum_{x} cnt[x] * cnt[x + k]$$

Tuy nhiên giá trị x có thể âm (|x|<=1000) ⇒ Cần ánh xạ, tức là với mỗi giá trị mảng a của đề bài, ta đều + 1000, để đảm bảo tất cả đều dương

Code mẫu

```
for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    cnt[x + MAX]++;// để tránh số âm, vì a_i trong [-1000, 1000]
}

long long ans = 0;
for (int x = -1000; x <= 1000; x++) {
    int y = x + k;
    if (y < -1000 || y > 1000) continue;
    ans += 1LL * cnt[x + MAX] * cnt[y + MAX];
}

cout << ans << "\n";
```

Chuyển vị ma trận

Ý tưởng: Lập bảng mới (bảng chứa kết quả) - mảng ans
Với mảng a có kích thước m*n thì mảng ans có kích thước n*m

⇒ Xử lý: For(int i = 1 ; i <= m ; ++i)
For(int j = 1 ; j <= n ; ++j)
ans[j][i] = a[i][j]

⇒ Cout ra mảng ans

Nhân ma trận

Cách làm:
Lập mảng ans
với phần tử ans[i][j] ⇒
For (int k = 1 ; k <= n ; ++k) ans[i][j]+=a[i][k]*a[k][j];

Sơn Cột

Ý tưởng

Ta có một ma trận chiều cao h[i][j]
Mỗi ô (i,j) là một cột khối lập phương chồng lên nhau.
Cần tính tổng diện tích bề mặt nhìn thấy.

Cách làm:

1. Gán tất cả ô ngoài biên có chiều cao = 0 (tưởng tượng có viền 0 bao quanh)
2. Với mỗi ô (i,j) xét 4 hướng: trên, dưới, trái, phải.
- Nếu hàng xóm có chiều cao thấp hơn, ta phải sơn phần chênh lệch.
- Nếu cao hơn hoặc bằng thì không cần sơn phía đó.

3. Sau khi cộng hết 4 hướng, cộng thêm **2 diện tích mặt đáy + mặt trên** cho mỗi cột (tức là +2 với mỗi ô có cột).

Code mẫu:

```cpp
long long Print(int i, int j) {
    long long res = 0;
    // 4 hướng: trên, dưới, trái, phải
    int dx[4] = {-1, 1, 0, 0};
    int dy[4] = {0, 0, -1, 1};

    for (int d = 0; d < 4; d++) {
        int ni = i + dx[d], nj = j + dy[d];
        long long neighbor = a[ni][nj];
        if (a[i][j] > neighbor) {
            res += a[i][j] - neighbor;
        }
    }
    return res;
}
```

```cpp
long long ans = 0 ;
for (int i = 1; i <= m; i++) {
    for (int j = 1; j <= n; j++)
        ans += Print(i, j);
        ans += 2; // mặt trên và mặt đáy
    }
}
```

Biến đổi bảng số

Gọi mảng C với C[i][j] = 1 nếu a[i][j] == b[i][j], = 0 nếu ngược lại

⇒ Mục đích của chúng ta cần biến mảng C về thành mảng 0 (tất cả các phần tử trong mảng = 0)

⇒ Cách làm: Vì một hàng hoặc một cột chỉ nên “Biến đổi” nhiều nhất 1 lần, bởi vì nếu “Biến đổi” 2 lần thì thà rằng không biến đổi còn hơn

⇒ Chúng ta cần tìm số phép biến đổi ít nhất ⇒ Tất nhiên ở bài này chúng ta không thể chạy “trâu” o(2^n) được

⇒ Hãy nhớ rằng: một hàng hoặc một cột chỉ được biến đổi nhiều nhất 1 lần ⇒ Do vậy, ta thử biến đổi hoặc không biến đổi cột 1 trước ⇒ Xác định được các hàng cần đổi ⇒ Từ các hàng cần đổi thì ta xác định được các cột cần đổi tiếp theo

8/9/2025

Sắp xếp chẵn lẻ

Các bạn tách các số chẵn và số lẻ ra mảng riêng, sau đó sort lại 2 mảng riêng đó.
Sau đó cout ra mảng chẵn trước theo thứ tự tăng dần, và cout mảng lẻ theo thứ tự giảm dần
Code mẫu:

```
vector <int>chan,le;
cin >> n ;
for (int i = 1 ; i <= n ; ++i) cin >> a[i];
for (int i = 1 ; i <= n ; ++i)
{
    if(a[i]%2==0) chan.pb(a[i]);
    else le.pb(a[i]);
}
sort(chan.begin(),chan.end());
sort(le.begin(),le.end());
for(int i = 0 ; i < chan.size(); ++i) cout << chan[i] << ' ' ;
for(int i = le.size()-1; i >=0 ; --i) cout << le[i] << ' ' ;
```

Trong đoạn code trên, mình có sử dụng mảng động (vector), các bạn có thể đọc 2 nguồn tham khảo ở đây Sorting a Vector in C++ và Vector in C++ STL

Sắp xếp ổn định

Ở bài này, các bạn nên tạo 1 mảng chỉ số và sort lại theo điều kiện mới (2 chỉ số chỉ giá trị khác nhau thì sẽ bị ưu tiên chỉ số chỉ số nhỏ hơn, 2 chỉ số chỉ giá trị bằng nhau thì sẽ ưu tiên số đứng trước)

Các bạn lưu ý rằng, ở đây mình sort trên mảng stt theo điều kiện mới

Code mẫu:

```
bool comparenew(const int&x , const int&y)
{
    if(a[x]==a[y]) return x<y;
    return a[x]<a[y];
}

void Do()
{
    cin >> n ;
    for (int i = 1 ; i <= n ; ++i)
    {
        cin >> a[i];
        stt[i] = i;
    }
    sort(stt+1,stt+1+n,comparenew);
    for (int i = 1 ; i <= n ; ++i) cout << stt[i] << ' ' ;
}
```

Lịch sửa chữa oto

Đặt c[i] = a[i]/b[i]
Để ý rằng ta sẽ ưu tiên xe thứ i với c[i] lớn nhất
***** Chứng minh:**

Giả sử rằng ta sẽ ưu tiên xe c[**j**] (với c[**j**] < c[**i**]) hơn xe **i** - Tức là xe **j** sẽ được sửa trước xe **i**
Đặt t là khoảng thời gian lúc bắt đầu sửa chiếc xe thứ j
⇒ Tổng thời gian sửa cả 2 chiếc xe, với xe thứ j được sửa trước là
T1 = (t+b[j])*a[j] + (t+b[j]+b[i])*a[i]
= t*a[j]+b[j]*a[j] + t*a[i] + b[j]*a[i]+b[i]*a[i]

Giả sử ngược lại, ta sẽ ưu tiên xe i (với c[i]>c[j]) hơn xe j - Tức là xe i sẽ được sửa trước xe j
Đặt t là khoảng thời gian lúc bắt đầu sửa chiếc thứ i
⇒ Tổng thời gian sửa cả 2 chiếc xe với xe thứ i được sửa trước là
T2 = (t+b[i])*a[i] + (t+b[j]+b[i])*a[j]
= t*a[i]+b[i]*a[i]+t*a[j]+b[j]*a[j]+b[i]*a[j]

** **So sánh T1 và T2**
T1-T2 = b[j]*a[i]-b[i]*a[j]
Ta để ý rằng c[i] > c[j] ⇒ a[i]/b[i] > a[j]/b[j] ⇒ a[i]*b[j] - b[i]*a[j] > 0
⇒ T1-T2 > 0 ⇒ T1 > T2

Vậy ta thấy rằng: Việc ưu tiên chiếc xe có giá trị c lớn hơn sẽ giúp tối ưu thời gian hơn

*** **Cách cài**
Tương tự bài **Sắp xếp ổn định,** mình sẽ lập mảng stt, và sort theo điều kiện mới

Code mẫu:

```
bool comparenew (const int& i, const int&j)
{
    return a[i]*b[j] > b[i]*a[j];
    /* Ưu tiên c[i] > c[j] hơn nhưng mình không ghi
    a[i]/b[i]>a[j]/b[j] vì phép chia này có thể xảy ra sai số */
}

void Do()
{
    cin >> n ;
    for (int i = 1 ; i <= n ; ++i)
    {
        cin >> a[i];
        stt[i] = i;
    }
    for (int i = 1 ; i <= n ; ++i) cin >> b[i];
    sort(stt+1,stt+1+n,comparenew);
    long long t = 0, ans = 0;
    for (int i = 1 ; i <= n ; ++i)
    {
        int id = stt[i]; // Sau khi sort, stt[i] không bằng i nữa
        t+=b[id];
        ans += t*a[id];
    }
    cout << ans;
}
```

Sắp xếp số

Bài này mình phải sort lại các số theo điều kiện sau:
Lưu ý rằng: Ở bài này các bạn hoàn toàn có thể sử dụng string và code hàm compare rất dễ dàng, nhưng hãy sử dụng trên kiểu int để vận dụng sự khéo léo của mình 😀
Khi so sánh 2 số a và b
Mình sẽ so sánh số ab hay ba lớn hơn, nếu ab lớn hơn ba thì mình ưu tiên số a hơn, nếu ba lớn hơn thì ưu tiên số b hơn
Ví dụ khi so sánh số 9 với 96 thì ưu tiên số 9 hơn, khi so sánh số 9 với 224 thì cũng ưu tiên số 9 hơn, khi so sánh số 96 với số 22 thì ưu tiên số 96 hơn

Sau đó cout ra mảng sau khi sắp xếp (Bởi vì đã sort lại theo độ ưu tiên, nên khi cout ra thì mặc định đó là số lớn nhất mà không cần xử lý gì thêm)
Code mẫu:

```
bool comparenew ( const ll &x , const ll &y)
{
    int lx = Leng(x), ly = Leng(y);
    // Hàm Leng là hàm tìm độ dài của số
    // Các bạn tự code lại hàm Leng nhé
    ll xy  = x*pow(10,ly)+y, yx = y*pow(10,lx)+x;
    return xy > yx ;
}

void Do()
{
    cin >> n ;
    for (int i = 1 ; i <= n ; ++i) cin >> a[i];
    sort(a+1,a+1+n,comparenew);
    for (int i = 1 ; i <= n ; ++i) cout << a[i];

}
```

25/8/2025

Thuê xe đạp

sort lại mảng w
⇒ Vì 1 xe đạp chỉ chở tối đa 2 người, nên mình sẽ sắp xếp tham lam (Tức ông lớn nhất sẽ xếp với ông nhỏ nhất)
Nếu ông lớn nhất cộng với ông nhỏ nhất không vượt quá m thì ta ghép cặp 2 ông đó ⇒ và xét tiếp ông lớn nhất và nhỏ nhất còn lại tiếp theo
Nhưng nếu không xếp được thì ta xếp riêng ông lớn nhất vào 1 xe và xét ông lớn nhất còn lại tiếp theo

Đoạn con có tổng bằng 0

Giả sử ta có ps[i] là prefix sum (tổng cộng dồn từ 1 tới i) ở vị trí i. Tức ps[i] = a[1]+a[2]+....+a[i]

⇒ Tổng Đoạn con từ bắt đầu ở vị trí i và kết thúc tại j = ps[j]-ps[i-1] ⇒ Để cho tổng đoạn con dương ⇔ ps[j]-ps[i-1] = 0
⇔ ps[j] = ps[i-1]

Dùng mảng pair a
Với **a[i].first = ps[i] và a[i].second = i**
Khi sort lại mảng pair thì sẽ được ưu tiên sort theo phần tử đầu tiên (first)
⇒ Sau khi sort mảng a, các phần tử được sắp theo thứ tự tăng dần của giá trị first
⇒ Lúc này Bài này lại trở thành việc tìm cặp (x,y) sao cho a[x].first = a[y].first và a[y].second - a[x].second đạt max

⇒ Vì Các ông có giá trị first = nhau thì sẽ đứng cạnh nhau, và ông có giá trị second nhỏ hơn sẽ đứng trước. Do vậy với một đoạn các phần tử bằng nhau của mảng a thì ta sẽ đặt x là phần tử đầu tiên có giá trị đó và y là phần tử cuối cùng có giá trị sđó

⇒ Khi này ta dùng thuật toán 2 con trỏ để xử lý
Code mẫu xử lý đoạn nhỏ này:

```
for (int i = 1 ; i <= n ; ++i)
    {
        int j = i ;
```

```
        while(a[j+1].first == a[i].first) j++;
        res = max(res,j-i+1);
        i=j;
    }
```

Đoạn có tổng dương

Giả sử ta có ps[i] là prefix sum (tổng cộng dồn từ 1 tới i) ở vị trí i. Tức ps[i] = a[1]+a[2]+....+a[i]

⇒ Tổng Đoạn con từ bắt đầu ở vị trí i và kết thúc tại j = ps[j]-ps[i-1] ⇒ Để cho tổng đoạn con dương ⇔ ps[j]-ps[i-1] > 0
⇔ ps[j] > ps[i-1]

Dùng mảng pair a
Với **a[i].first = ps[i] và a[i].second = i**
Khi sort lại mảng pair thì sẽ được ưu tiên sort theo phần tử đầu tiên (first)
⇒ Sau khi sort mảng a, các phần tử được sắp theo thứ tự tăng dần của giá trị first
⇒ Khi này giá trị a[i].first (prefix sum) đã được sort theo thứ tự tăng dần, Tuy nhiên giá trị a[i].second lúc này có thể không = i nữa, do đã bị đảo lên
⇒ Lúc này bài trở thành tìm cặp phần tử (x,y) sao cho y>x và a[y].second-a[x].second đạt max

⇒ Cách làm: Gọi mảng Min với Min[i] = vị trí của min(a[1].second,a[2].second,....,a[i].second)

⇒ Với mỗi vị trí i ta đều thử xem a[i].second - a[Min[i-1]].second có lớn hơn res không, nếu có thì ta đặt res = a[i].second - a[Min[i-1]].second và dùng thêm 2 biến để lưu trữ điểm đầu và cuối của đoạn con đó

18/8/2025

Bài cặp số bằng nhau:

Giả sử ta có mảng cnt với cnt[value]: số lần xuất của giá trị value trong dãy đã cho
⇒ Với giá trị value thì ta có được cnt[value]*(cnt[value]-1)/2
Bởi vì giá trị của mảng a là nguyên dương và không vượt quá được 1e3, nên ta hoàn toàn có thể for tất cả các giá trị để tính

Bài Cặp số nguyên tố cùng nhau

Thuật toán: For 2 vòng để thử hết tất cả các cặp
Với mỗi cặp, thử check ước số chung lớn nhất, nếu ước số chung lớn nhất = 1 thì res++
Cuối cùng, sau khi for 2 vòng, thử hết tất cả các cặp thì cout << res

Bài Tích lớn nhất

Sort lại dãy theo thứ tự tăng dần
a1,a2,....,an
a1 là số nhỏ nhất và an là số lớn nhất
Lúc này tích lớn nhất là tích lớn nhất của 3 cặp a1*a2, an*a(n-1), a1*an (Bởi vì giá trị trong mảng a có thể âm)

Bài Kiểm tra Hoán vị

Giả sử ta có mảng per, với per[x] = true hoặc false (true nếu giá trị x đã xuất hiện, false nếu giá trị x chưa xuất hiện)
⇒ Nhập Input
cin >> p[i]
if(p[i] > n) ⇒ cout << "NO"; (do nếu dãy đó tồn tại 1 số lớn hơn số lượng phần tử thì chắc chắn đó ko phải hoán vị)
else if(per[p[i]]==true) ⇒ cout << "NO"; (bởi vì hoán vị không thể xuất hiện 2 số giống nhau)
else per[p[i]] ++; // Phép tăng per[p[i]] lên 1 đơn vị
Bài Tráo Bài (cardsez)
ĐỀ BÀI BẢO GÌ LÀM ĐÓ!!
Làm theo ý tưởng tự nhiên của đề bài
Với mỗi truy vấn (i,m,j) thì ta nhấc đoạn từ vị trí i tới i+m-1 ra **mảng mới**. Sau đó các phần tử ở vị trí i+m,i+m+1,..n sẽ nhảy lên vị trí i,i+1,...n-m (Tức từ tất cả ông từ vị thứ i+m tới ông thứ n, sẽ được tiến lên m bước).
Sau bước đó, ta thừa ra m phần tử ở cuối mảng ((bởi vì các ông ở cuối mảng đã dịch lên m vị trí)
⇒ Tiếp theo cần nhét m ông ở vừa được rút ra lúc trước vào vị trí j
Lúc này cần đẩy ông tử vị trí j,j+1,....,n-m xuống m đơn vị (Tức ông ở vị trí j thành j+m, j+1 thành j+1+m, … , n-m thành vị trí n)
⇒ Sau đó chỉ cần chèn lại m phần tử từ mảng mới vào mảng gốc ở vị trí j của mảng gốc
⇒ Mỗi truy vấn lặp lại như vậy và cuối cùng cout ra mảng
⇒ Độ phức tạp o(n*x)

Bài Tráo bài (Shuffle)

Bài này giống bài trước về nội dung, tuy nhiên về phần giới hạn của bài toán thì lại khác rất nhiều.

Nếu chạy giống thuật toán trước với độ phức tạp o(n*x) tương đương khoảng 1e6*1e5 = 1e11 ⇒ Chắc chắn không qua, do máy tính chỉ chạy được khoảng 5e7 trong 1s

Ta để ý rằng: Đề bài chỉ hỏi vị trí của k ông đầu tiên của mảng

⇒ Ý tưởng chung: LÀM NGƯỢC LẠI

Ví dụ:

Sau khi đổi chỗ xong truy vấn thứ x

⇒ Gọi Ông đứng ở vị trí 1 là b (ta đang cần tìm giá trị b)

⇒ Vậy ta có biết được vị trí của b sau truy vấn thứ x-1 không? ⇒ Biết được vị trí của b sau truy vấn thứ x-2?

Truy vấn thứ x-3,.....,2,1

⇒ Sau khi biết được vị trí của b lúc đầu thì ta biết được chắc chắn rằng giá trị của b bằng vị trí của b lúc đầu

⇒ NHIỆM VỤ BÂY GIỜ LÀ: CẦN LÀM HÀM UNDO

Ta tìm lần lượt giá trị của k số đầu tiên

⇒ Xét vị trí cur (current) vị trí sau truy vấn thứ x

- Mỗi truy vấn có i,m,j

Th1: Nếu j<=cur<=j+m-1 ⇒ Nằm trong vùng được di chuyển ⇒ vị trí của cur lúc trước là cur - j + i

TH2: Nếu i<=cur < j ⇒ Nằm trong vùng bị dồn toa lên ⇒ vị trí của cur lúc trước là cur + m

TH3: Nếu j+m-1<cur<=i+m-1 ⇒ Nằm trong vùng bị đẩy ra sau

⇒ Undo lần 1 ⇒ Vị trí cur sau truy vấn thứ x-1

⇒ … ⇒ Undo x lần

Code mẫu:

```
int n , k , q ;
struct T
{
    int i , m , j;
};
vector<T> v;
void Readinput()
{
    cin >> n >> k >> q ;
    for (int i = 1; i <= q ; ++i)
    {
        int x , m , j ; cin >> x >> m >> j ;
        v.pb({x,m,j});
    }
    reverse(v.begin(),v.end());
    for (int i = 1 ; i <= k ; ++i)
    {
        int cur = i;
        for (auto x:v)
        {
            if(x.j<=cur&&cur<=x.j+x.m-1) cur = cur - x.j + x.i ;
            else if (cur<x.j&&cur>=x.i) cur += x.m;
            else if (cur>x.j+x.m-1&&cur<=x.i+x.m-1) cur -= x.m;
        }
        cout << cur << ' ' ;
    }
}
```

Bài Chia đoạn

Thuật toán: CHẶT NHỊ PHÂN KẾT QUẢ

Tìm kiếm nhi phân | VNOI Wiki (Nguồn tài liệu cho các bạn)

Hàm check cần kiểm tra xem n thanh gỗ có chia được số lượng >= k với độ dài = mid hay không

Nếu hàm check(mid) return true ⇒ Hoàn toàn có thể chia được >= k đoạn với độ dài < mid ⇒ L = mid+1

Nếu hàm check(mid) return false ⇒ Chắc chắn không thể chia được >= k đoạn với độ dài < mid ⇒ R = mid-1

Cuối cùng, kết quả của bài là R

11/8/2025

Bài Biến đổi số:

Không mất tính tổng quát, giả sử a < b < c

Yêu cầu 1: Tính số biến đổi ít nhất

⇒ Nhận xét.

TH1: Nếu khoảng trống giữa a ⇒ b hoặc b⇒c bằng 1 thì số biến đổi ít nhất là 1

Ví dụ 2 4 10 ⇒ Biến thằng 10 thành 3 ⇒ 2 3 4

TH2: Nếu khoảng trống giữa a ⇒ b hoặc b⇒c đều khác 1 ⇒ số lượng khoảng trống giữa a⇒b và b⇒c sẽ >= 2

Vì a < b < c Nên biến đổi c thành a + 2

⇒ 3 số lúc này biến đổi thành a < a + 2 < c ⇒ Thành TH1 với số lượng khoảng trống giữa a và a+2 là 1

Yêu cầu 2: Tính số biến đổi nhiều nhất

⇒ Gọi m là số lượng ô trống giữa a ⇒ b và n là số lượng ô trống giữa b ⇒ c

⇒ m = b - a - 1 và n = c - b - 1

Kết quả = max(m,n)

Do, giả sử m > n

⇒ Bước 1: đặt c = b-1 ⇒ 3 số thành a < b-1 < b

⇒ Bước 2: đặt b = b-2 ⇒ 3 số thành a < b-2 < b-1

….

⇒Bước m: 3 số thành a < a+1 < a+2

Bài Số chính phương:

Vì luôn có cách biểu diễn 1 số thành tích của các số nguyên tố
Giả sử

$n = 2^{a_i}3^{a_2}5^{a_3}$....

$m = 2^{b_1}3^{b_2}5^{b_3}$....

Nếu tìm được số m chia hết cho n thì các số mũ b1>a1 b2>a2 b3>a3,,.... và b1,b2,b3 phải là số chẵn
⇒ Ý tưởng: Phân tích thành các số nguyên tố, tìm số mũ và xử lý

Bài Tìm phần tử

Cho 3 số n, m, t
Gọi d = n%m
⇒ TH0: Số lượng số chia hết cho m là n/m
⇒ TH1: số lượng số chia m dư 1 là n/m+1
⇒ …..
⇒ THd: Số lượng số chia m dư d là n/m+1
⇒ THd+1: Số lượng số chia m dư d+1 là n/m
⇒ ….
⇒ THm-1: Số lượng số chia m dư m-1 là n/m

⇒ Ta nhận thấy rằng: Với số dư từ 1⇒d thì có n/m+1 số. Với số dư từ 0 và từ d+1⇒n thì có n/m số
⇒ Cần tìm t thuộc số dư nào

⇒ **Nếu t nhảy vào số dư x thì cần tìm thứ tự của t trong số dư x đó.**
Ví dụ với test đề bài: 10 4 8 thì số t = 8 là số thứ 3 trong số dư 2
⇒ Kết quả = (Số thứ tự-1)*m+x
Nhưng nếu t ở TH0 ⇒ Kết quả = Số thứ tự*m

4/8/2025

Bài 1: Phương trình tuyến tính thuần nhất 1 ẩn

Dễ thấy, nếu a==0 và b==0 thì hệ vô số nghiệm
nếu a==0 và b!=0 thì hệ vô nghiệm
Nếu a!=0 và b!=0 thì hệ có nghiệm là -b/a

Bài 2: Tìm số

Với n là bất kì số chẵn hoặc số lẻ thì sẽ có (n+1)/2 số lẻ ở đầu, n/2 số chẵn ở cuối
⇒ Nếu k <= (n+1)/2 thì đó là số lẻ ⇒ ans = 2*k - 1 ;
⇒ Nêu k > (n+1)/2 thì đó là số chẵn ⇒ ans = 2*(k-(n+1)/2);

Bài 3: Tính diện tích

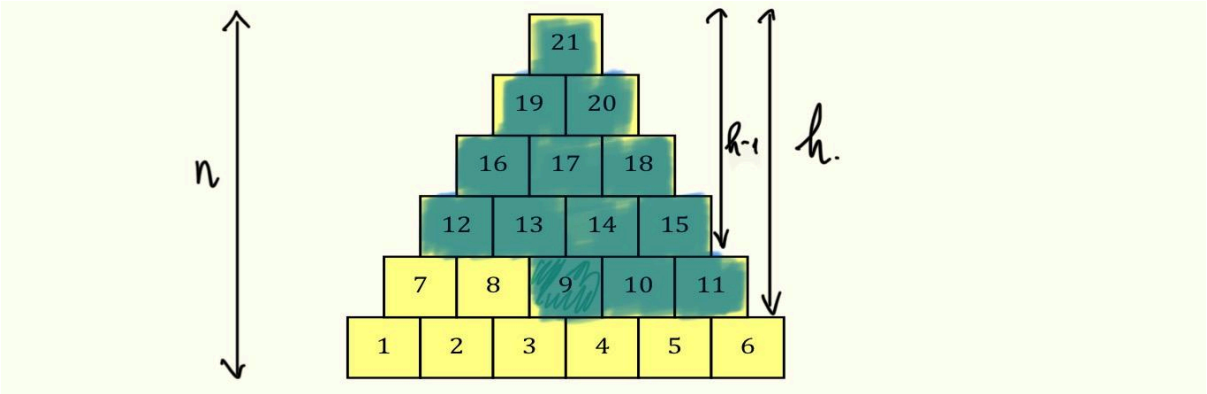
Dựa vào công thức heron, công thức tính đường tròn ngoại tiếp, công thức tính đường tròn nội tiếp, thì ta sẽ tính ra lần lượt được V,S,R

Bài 4: Đếm điểm

Góc dưới trái là (x1,y1)
Góc trên phải là (x2,y2)
Cho một điểm (x,y) bất kì, nếu x1<=x<=x2 và y1<=y<=y2 thì điểm (x,y) sẽ thuộc trong đoạn đã cho
⇒ Kiểm tra lần lượt từng điểm (x,y)
if(x1<=x&&x<=x2&&y1<=y&&y<=y2) cnt++;

Bài 5: Số tầng

Số phòng từ tầng cao nhất tới hết tầng thứ m trong tòa n tầng là
$1 + 2 + \ldots + (n - m + 1)$
Đặt h = n-m+1
$=> 1 + 2 + .. + h = h * (h + 1)/2$
Rõ ràng, nếu phòng thứ i ở tầng m thì tổng số lượng phòng từ phòng **thứ i tới phòng n*(n+1)/2** phải nhỏ hơn hoặc bằng h*(h+1)/2 và lớn hơn (h-1)*h/2
⇒ Đặt x = số lượng phòng từ phòng thứ i tới phòng n*(n+1)/2 = n*(n+1)/2 - i + 1



Với ví dụ với ô số 9 ở trên, số lượng ô tô đậm là giá trị của x

$\Leftrightarrow h * (h-1)/2 \leq x \leq h * (h+1)/2$

$\Leftrightarrow h * (h-1) \leq 2 * x \leq h * (h+1)$

$\Leftrightarrow \sqrt{h * (h-1)} \leq \sqrt{2 * x} \leq \sqrt{h * (h+1)}$

Lưu ý rằng các phép toán ở đây là ép kiểu xuống. Nên ta có được phương trình

$\Leftrightarrow h - 1 \leq \sqrt{2 * x} \leq h$

⇒ Lúc này chỉ có 2 trường hợp xảy ra là i có thể đứng ở tầng h và h-1 từ trên xuống, tức là tầng n-h+1 hoặc n-(h-1)+1

⇒ Kiểm tra lại điều kiện để xác định

Code mẫu:

```
ll CntInverse ( ll i )
{
    return n*(n+1)/2-i+1;
}

void Do()
{
    cin >> n >> k;
    while(k--)
    {
        cin >> i ;
        ll x = CntInverse(i);
        ll h = sqrt(2*x);
        if(h*(h+1)/2 < x) h++;
        cout << n-h+1 << '\n';
    }

}
```

Bài 6: Tìm X

Gọi v là số nguyên sao cho

a+v chia hết cho b và b+v chia hết cho a

⇒ số v có thể là lcm(a,b) - a - b
Do $v = lcm(a,b) - a - b$
a+v = $lcm(a,b) - b$ chia hết cho b, do lcm(a,b) và cả b đều chia hết cho b
Chứng minh tương tự với b+v

Nếu v > 0 ⇒ v chính là số thỏa mãn, nguyên dương và nhỏ nhất
Nếu v < 0 ⇒ thì cần cộng thêm lcm(a,b) vào v để v thành số nguyên dương

Bài 7: Cặp số

Dễ thấy, nếu n là số chẵn thì có LCM nhỏ nhất là n/2
Và nếu n là số lẻ thì có LCM lớn nhất là n/2*(n/2+1)
⇒ Cần giải quyết TH LCM lớn nhất của n chẵn và LCM nhỏ nhất của n lẻ
TH1: LCM lớn nhất của n chẵn
⇒ Dễ nhận thấy LCM(a,b) = (a*b)/gcd(a,b)
⇒ LCM(a,b) max ⇔ a*b max và gcd(a,b) min ⇔ gcd(a,b) == 1 và b-a min
⇒ for i từ n/2, nếu gcd(i,n-i) == 1 lcm(i,n-i) max

TH2: LCM nhỏ nhất của n lẻ
Vì LCM(1,n-1) = n-1 ⇒ Nếu tìm được LCM nhỏ nhất thì sẽ không vượt quá n-1

Do vậy nếu LCM(a,b) = (a*b)/gcd(a,b)
Nếu gcd(a,b) != a và gcd(a,b) != b thì a*b/gcd(a,b) > n-1

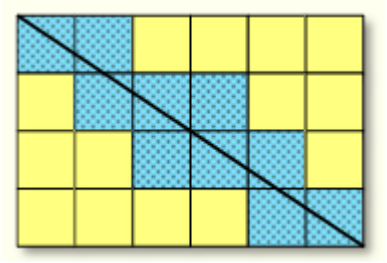
⇒ **Các bạn thử chứng minh!**
Do vậy ta xác định được rằng a hoặc b sẽ phải = gcd(a,b)
⇒ Giả sử a = gcd(a,b) ⇒ b chia hết cho a
⇒ for các cặp như vậy

```
for (int i = 2;i <= sqrt(n);i++){
        if (n % i == 0){
            min_lcm = min(min_lcm,n-i);
            min_lcm = min(min_lcm,n - (n/i));
        }
    }
```

Bài 8: Đếm ô

Khi vẽ đường thẳng từ góc trên trái đến góc dưới phải:
Mỗi khi nó đi qua một đường kẻ dọc ⇒ vào một ô mới theo chiều ngang.
Mỗi khi nó đi qua một đường kẻ ngang ⇒ vào một ô mới theo chiều dọc.
Tuy nhiên, nếu đường đi qua **một giao điểm lưới** (cả đường dọc và ngang) thì ta đã tính trùng ô tại đó.

Đoạn từ (0,0) tới (m,n) cắt (chạm) mỗi **đường thẳng dọc** m-1 lần
Tương tự cắt **đường thẳng ngang** là n-1 lần
Khi đoạn đi qua **giao điểm lưới** (tức là cắt tại tọa độ (x,y) thì m/x = n/y và là số nguyên)
⇒ Ví dụ giao điểm lưới (2,3) như trong ví dụ đề bài



⇒ Ta đã đếm trùng hai lần (dọc và ngang). Số giao điểm lưới nội bộ như vậy bằng gcd(m,n)−1

Tổng ô được đếm = (m-1)+(n-1)+(gcd(m,n)-1)+1 = m+n+gcd(m,n)+2
(cộng 1 vì đếm cả ô chứa điểm đầu)

Bài 9: Tìm số

Hướng làm: Bao hàm loại trừ + Chặt nhị phân kết quả
Chặt nhị phân giá trị x cần tìm
⇒ Số thứ tự của x nằm trong dãy của đề bài là x/a + x/b - x/lcm(a,b)
Đặt L = 1 , R = 1e9

```
while(L<=R)
{
   int mid = (L+R)/2;
   if(Stt(mid) >= n) r = mid-1; // Hàm Stt sẽ tính số thứ tự của giá trị mid trong dãy
   else l = mid+1;
}
cout << l;
```

31/7/2025

Bài 1: Cộng dồn

Đơn giản nếu số a < b thì ta lấy a += b

⇒ bởi vì muốn tăng nhanh nhất thì phải cộng lớn hơn

⇒ Cách cài:

```
while (a<=n || b<=n){
        if (a<b){
            a += b;
        } else{
            b += a;
        }
        c+=1;
    }
    cout << c << "\n";
```

Bài 2: số siêu nguyên tố

Hàm check số nguyên tố

```
bool check(long long x)
{
    if (x<2) return false;
    for(long long i=2; i*i<=x; i++)
    if (x%i==0) return false;
    return true;
}
```

Với việc check xem số x có phải số siêu nguyên tố hay ko, ta bỏ dần các phần tử bên phải của nó và lại check nguyên tố

Bài 3: Cắt giấy

Với mảng giấy có độ dài m*n (m<=n)

Thì ta có được số lượng mảng giấy hình vuông được cắt với kích thước m*m là n/m
Và mảng giấy còn lại có kích thước là m*(m%n)

Làm như vậy cho đến khi m%n==0

28/7/2025

Buổi này các bạn ôn tập về đệ quy

Bài 2: Lũy thừa

- Dùng đệ quy để giải quyết
- Khi cần tính a^k thì ta sẽ gọi đệ quy để tính $a^{k/2}$ trước. Đó là cách đệ quy chạy. Ở trường hợp lớn, sẽ quy về trường hợp nhỏ hơn để giải quyết. Khi có kết quả của trường hợp nhỏ thì sẽ giải trường hợp lớn hơn.
- Tất nhiên khi k = 0 thì $a^k = 1$
- Code mẫu

```
long long p(long long a,long long n, long long m){
    if (n==0) return 1;
    long long t = p(a, n/2, m);
    if(n%2==0) return (t) *(t) %m;
    else return (t*t%m)*a%m;
}

int main()
{
    long long a, n, m;
     long long kq = 1;
    cin >> a >> n >> m;


    cout << p(a, n, m);

}
```

Bài 3: Phép tính lũy thừa

Tính $a^b * b^a \% m$

Việc tính a^b và b^a sẽ dùng hàm p ở bài trước để làm. Nhưng m <= 10^{18} nên ở mỗi phép mod, chỉ đảm bảo cho biểu thức a^b và b^a <= 10^{18} nên nếu để $a^b * b^a$ rồi sau đó mới mod thì sẽ xảy ra tràn số

⇒ Cần viết hàm để nhân $a^b * b^a$ *sao cho không bị tràn số*

⇒ Ý tưởng: Viết hàm Mul(x,y,m) // x*y %m

Khi lấy x*y, gọi đệ quy về x*y/2 để xử lý

Với y = 0 thì return 0

ll **mul**(ll x, ll y, ll m)

```
{
    if (y==0) return 0;
    ll t = mul(x,y/2,m);
    if(y%2==0) return (t+t)%m;
    else return (t+t+x)%m;
}
```

Bài 4: Tìm chữ số

a = a % b

Lặp liên tục (k - 1) lần:

    a = a*10^ (k - 1) % b;

Lần cuối lấy kết quả:x = a * 10/b

a = a %b

a = 99

a %= 140 = 99

Lần	a	b	KQ	dư
1	99 *10	140	7	10
2	10*10	140	0	100
3	100*10	140	7	20
4	20*10	140	1	60
5	60 * 10	140	4	40
6	40*10	140	2	120

Ta cần tính (a*10^(k-1)%b)*10/b

```cpp
#include<iostream>
using namespace std;
using ll = long long;
using ull = unsigned long long;
ull a, b, k;
ull Mul(ull a, ull b, ull m) //a * b % m
{
    if(b == 0) return 0;
    ull t = Mul(a, b/2, m);
    t = (t + t)% m;
    if(b % 2 == 0) return t;
    return (t + a)%m;


}
ull power(ull a, ull b, ull m)//a^ b% m
{
    if(b == 0) return 1;
   // if(b == 1) return a % m;
    ull t = power(a, b/2, m);
    t = Mul(t, t, m) ;
    if(b % 2 == 0) return t;
    return Mul(t , a , m);
}
int main()
{

    int t;
    cin >> t;
    while(t--)
    {
        cin >> a >> b >> k;
        a = a % b;
        a = Mul(a , power(10, k - 1, b) , b);
        cout << a * 10/b << endl;
    }
    /*
    ll a, b;
    a = 999999999999999999; b =  987654321987654321, k = 123456789123456789;
    cout << a * b % k;*/
}
```

Bài 5: So sánh phân số

Cách để so sánh phân số

- Quy đồng mẫu số
- Sau đó so sánh tử số

nếu tử số bằng nhau thì mẫu nào bé hơn thì phân số đó lớn hơn.

Không thể dùng int64 để quy đồng vì bị tràn số

Giải pháp 1: Xử lý số lớn (__int128)
Giải pháp 2:
x = a/b, y= c/d
so sánh x với y? -> không ổn khi chênh lệch giữa tử số và mẫu số quá lớn -> phép chia xuất hiện **sai số**.
Giải pháp 3:
ta so sánh hai phân số với 1:
Nếu a > b và c < d -> ta KL được phân số nào lớn hơn.

- Nếu cả hai phân số đều > 1: (tử số > mẫu số)

ta tính phần nguyên:
1234/4, 56789/2.
Trường hợp phần nguyên bằng nhau:
5/4 9/7
(1)¼
(1)2/7
Ta chỉ còn so sánh hai phân số 1/4 và 2/7. Phân số nào lớn hơn thì phân số ban đầu lớn hơn.

- tử số < mẫu số:

Không thể quy đồng vì tràn số:
a/b và c/d
=> đảo ngược hai phân số: b/a và d/c là hai phân số có tử lớn hơn mẫu
nếu b/a > d/c -> a/b < c/d

⇒ Vận dụng đệ quy để so sánh 2 số

24/7/2025

Bài 3: Chia hết

- Bài này các bạn sẽ dùng bao hàm loại trừ
- Các bạn nào hứng thú về phần kiến thức này có thể tham khảo tại đây: Bao hàm - Loại trừ (Inclusion-Exclusion) | VNOI Wiki
- Công thức của bài này là

n - n/2 - n/3 - n/5 - n/7 + n/6 + n/10 + n/14 + n/15 + n/21 + n/35 - n/30 - n/42 - n/70 - n/105 + n/210

Bài 4: Pha muối

giả sử b>=a

- Nhận xét nếu c không thuộc khoảng [a,b] thì kết quả là 0
- Gọi k1 là số bình của a, k2 là số bình của b

⇒ (k1+k2)*c = k1*a + k2*b
⇔ k1(a-c) + k2(b-c) = 0
⇔ k1(c-a) = k2(b-c)
⇔ k1/k2 = (b-c)/(c-a)
Vì mình đang cần tối ưu (k1+k2) (Tổng số lọ)
⇒ k1/k2 cần là nguyên số tối giản ⇔ gcd(k1,k2) = 1
⇔ k1=(b-c)/gcd(b-c,c-a), k2 = (c-a)/gcd(b-c,c-a)
⇒ k1 + k2 = ….

Bài 5: Tìm số

ý tưởng cần tìm số y lớn nhất có gcd(x,y) là lớn nhất

⇒ số x được phân tích ra thừa số nguyên tố có dạng

$x = 2^{x_0}3^{x_1}5^{x_2}$.....

Do vậy ta cần tìm số <u>nguyên tố **v** </u>nhỏ nhất mà có có $x_i > 0$

⇒ Lúc này gcd() lớn nhất có thể chính là x/v và số y lớn nhất có thể x/v*(v-1)

⇒ kết quả của bài là $(x/v)^2 * (v - 1)$

Bài 6: Đếm hình vuông

Với hình vuông kích thước n^2

Ta có hình vuông kích thước 1^2 là n^2

$Ta\ có\ hình\ vuông\ kích\ thước\ 2^2$ là $(n - 1)^2$

$Ta\ có\ hình\ vuông\ kích\ thước\ 3^2$ là $(n - 2)^2$

…

$Ta\ có\ hình\ vuông\ kích\ thước\ n^2$ là $(n - (n - 1))^2$

Do vậy kết quả của bài toán là

$1^2 + 2^2 + \ldots + (n - 2)^2 + (n - 1)^2 + n^2\ (1)$

Theo công thức tính tổng bình phương, ta có thể viết lại biểu thức 1 thành

$n * (n + 1) * (2n + 1)\ /6$

Vì khi mod sẽ không đảm bảo tính đúng đắn của biểu thức. Do vậy bài này ta cần tối giản biểu thức cho 6 trước khi mod. Hoặc dùng nghịch đảo modulo.

Bài 7: Cắt giấy (Đã viết solution ở bài 4 ngày 14/7/2025)

21/7/2025

Đóng gói

- Nếu có n bút chia vào các hộp (mỗi hộp chứa đc k bút)
  - Nếu n chia hết cho k => n/k hộp
  - Nếu n không chia hết cho k => Lấy lượng bút nhiều nhất mà chia được vào các hộp; dư để vào 1 hộp riêng
    => Làm tròn lên của n/k

Ví dụ: 7 bút chia vào các hộp chứa đc 3 cái => Cần (làm tròn lên của 7/3) = 3 cái

6 bút chia vào các hộp chứa đc 2 cái => 6/2 = 3

Cách cài đặt:

- Thư viện cmath có hàm ceil(x) => Làm tròn lên của x

```
cout << ceil(n / (long double)k);
```

- Tính chất toán học

```
#include <iostream>
#include <cmath>
using namespace std;

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

    cout << (n + k - 1) / k;
}
```

Quan hệ so sánh

- Lặp lại n lần
  - Nhập a, b
  - So sánh và in ra

```
int n;
cin >> n;
for (int i = 1; i <= n; ++i) {
	int a, b;
	cin >> a >> b;
	// In ra theo đề bảo
	…
	// In ra xuống dòng
	cout << “\n”;
}
```

For và while

```
int cnt = 1;
while (cnt <= 100) {
	cout << ‘#’;
	cnt = cnt + 1;
}
```

```
	cout << ‘#’;
}
```

```
for (int cnt = 1; cnt <= 100;
cnt = cnt + 1) {
```

Chữ số min max

- Lấy chữ số tận cùng: n % 10

- Bỏ chữ số tận cùng: n = n / 10
- n % 10
- n = n / 10
- n % 10
- n = n / 10
- ..

```
int minv = 1e9;
int maxv = -1;

while (n > 0) {
	minv = min(minv, n % 10);
	maxv = max(maxv, n % 10);

	n = n / 10;
}
```

Tính tổng

- Tổng đoạn [a, b]: (a+b) * (b-a+1) / 2
- Tổng các số chẵn:
  - a chẵn b chẵn: (a + b) * ((b - a) / 2 + 1) / 2
  - a chẵn b lẻ: (a + (b-1)) * (((b-1) - a)/2 + 1) / 2
  - a lẻ b chẵn: (a+1+b) * ((b-(a+1))/2 + 1) / 2
  - a lẻ b lẻ: ((a+1) + (b-1)) * (((b-1)-(a+1)) / 2 + 1) / 2

ZIGZAG

Nhận xét: Hàng thứ i sẽ chứa các số có giá trị từ (i - 1) * n + 1 → i * n

Ở bài này ta có 2 nhiệm vụ

Nhiệm vụ 1: Tìm giá trị ở ô (x,y)
Ta có số bắt đầu hàng i là base = (i - 1) * n + 1
Nếu hàng chẵn: ans = base + n - y (đánh số xuôi)
Nếu hàng lẻ: ans = base + y - 1 (đánh số ngược)
Nhiệm vụ 2: Tìm ô chứa giá trị z
Xác định hàng chứa ô z: row = (z - 1) / n + 1 (làm tròn lên của z / n)
Xác định số thứ tự của z trong hàng đó: pos = z % n (nếu z % n == 0 thì số thứ tự của z trong hàng là pos = n)
→ Cột chứa giá trị z:
Nếu hàng chẵn: col = n - pos + 1
Nếu hàng lẻ: col = pos

Khoảng cách số

Phân tích: 100 = 2 * 2 * **5** * 5
360 = 2 * 2 * **2** * **3** * **3** * 5
→ Số bước để chuyển từ 100 sang 360 chính là số phần tử khác nhau khi phân tích cả 2 số thành tích các thừa số nguyên tố
→ Ta có thể phân tích 2 số thành tích các thừa số nguyên tố rồi đếm số phần tử khác nhau
Để việc đếm dễ dàng hơn, ta có thể bỏ hết phần chung bằng cách chia cả 2 số cho gcd của chúng rồi đếm số phần tử khi phân tích sau khi chia

Phân số gần nhất

Ở đây ta cần tìm 2 số a và b sao cho a / b gần với x nhất, hay nói cách khác abs(x - a/b) là min
Ta có thể sử dụng kĩ thuật 2 con trỏ để giải quyết.

1 con trỏ đại diện cho a và 1 con trỏ đại diện cho b
Ban đầu cả 2 con trỏ bằng 1, nếu ta thấy abs(x - a/b) < ans thì ta cập nhật lại ans = abs(x - a/b) và lưu lại vị trí 2 con trỏ vừa tìm thấy.
Ở mỗi vòng lặp, nếu a/b < x thì ta tăng a lên, ngược lại ta tăng b, bằng cách này số ta xét đến sẽ luôn gần x nhất

```
long double x, ans = 1e18;
long long n, ans1, ans2;
cin >> x >> n;
long long a = 1, b = 1;
while(a <= n && b <= n)
{
    double res = (double)(a) / (double)(b);
    if(abs(x - res) < ans)
    {
        ans = abs(x - res);
        ans1 = a;
        ans2 = b;
    }
    if(res < x) a++;
    else if(res > x) b++;
}
cout << ans1 << " " << ans2;
```

Bộ ba

Vì a < b < c nên a < n / 3
Nếu ta cố định số a trước thì việc của ta là đếm số cặp thỏa mãn b + c = n - a và b < c
→ b <= (n - a - 1) / 2
mà b >= a + 1
→ Số cặp là (n - a - 1) / 2 - (a + 1) + 1
Ta for theo a
for(int a = 1; a < n / 3; a++)

```
{
	b = a + 1;
	x = (n - a - 1) / 2;
	if(b <= x) ans += x - a + 1;
}
cout << ans;
```

Số Fibonacci

Gọi
fib_i, fib_{i-1} là 2 số $fibbonaci$ đang xét, ta nhận thấy rằng

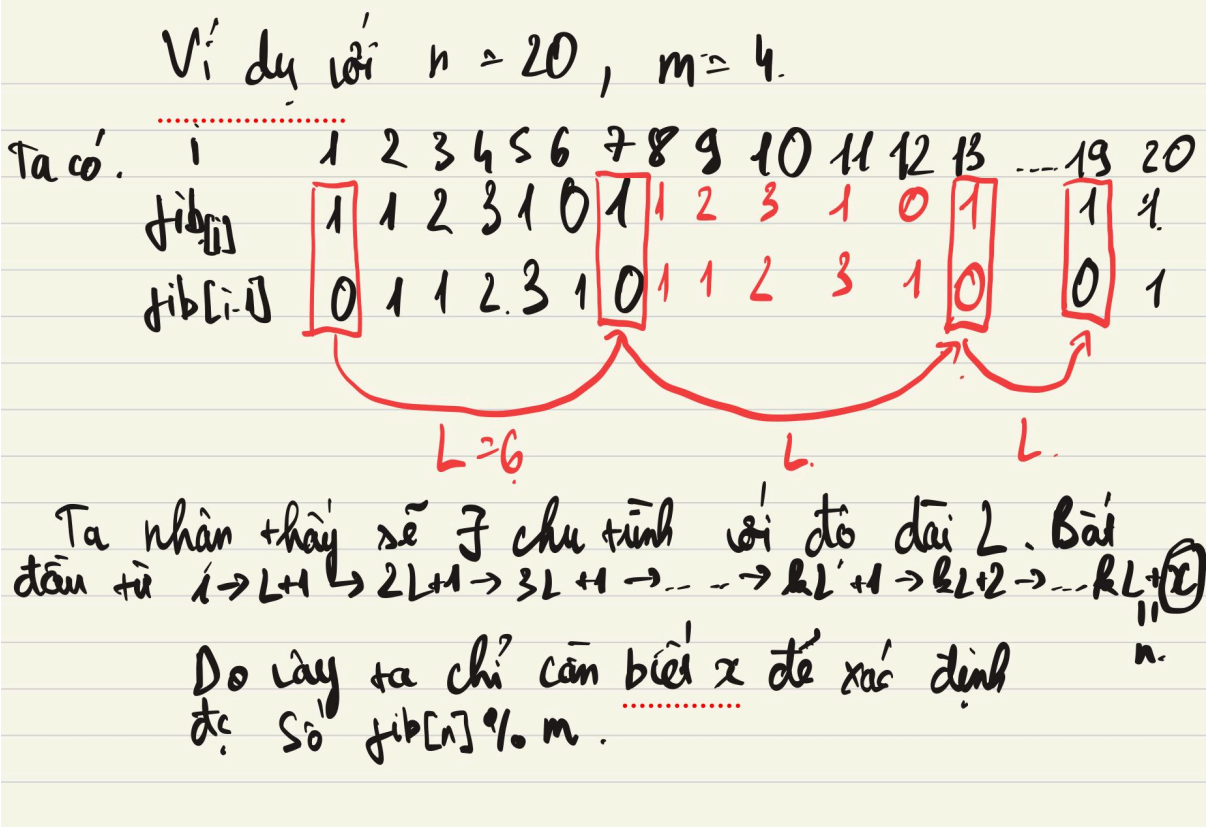
fib_i%m && fib_{i-1}%m <= m <= 10^3

⇒ các giá trị có thể của cặp số

(fib_i%m, fib_{i-1}%m) là hữu hạn (m^2) ≈ 10^6 nên sau một độ dài L các phép toán fibonacci nhất định thì (fib_i%m, fib_{i-1}%m) sẽ lặp lại giá trị đã được tính từ trước.

Ta gọi đó là chu trình có độ dài L tương ứng với n và m

Vì cứ sau một chu trình có độ dài L thì sẽ lại quay lại điểm (1,0) nên ta sẽ lấy độ dài của dãy fibonacci cần tính mod cho L ⇒ sẽ ra được đoạn thừa ra ⇒ do vậy ta có thể tính được giá trị đề bài yêu cầu



Sơn hình vuông

Ta có thể tính xem ô (i, j) ở lớp nào
Công thức tính lớp của ô (i, j)
layer = min(i - 1, j - 1, n - i, n - j) + 1
Nếu layer % 3 = 0 → vàng
layer % 3 = 1 → đỏ
layer % 3 = 2 → xanh

17/7/2025

Bài 1: Đường tàu (Metro)

Viết hàm tính độ dài theo chiều kim đồng hồ (1) với d(s,t) là độ dài từ s⇒t theo chiều kim đồng hồ

⇒ Tính độ dài đường đi từ a⇒b (đường đi từ An tới Bình) ⇒ gọi là dab
⇒ Tính độ dài đường đi từ a⇒x (đường đi từ An tới đích của mình) ⇒ gọi là dax
⇒ Tính độ dài đường đi từ b⇒y (đường đi từ Bình tới đích của mình) ⇒ gọi là dyb (vì (1) và b⇒y là ngược chiều kim đồng hồ nên muốn tính khoảng cách từ b⇒y ta phải xét ngược lại là d(y,b))
Nếu An và Bình có thể gặp nhau, thì:
TH1: An và Bình gặp nhau khi tàu An chưa vượt qua chỗ Bình và tàu Bình chưa vượt qua chỗ An (gặp nhau khi chưa lướt qua nhau)

- Đồ dài đường đi giữa a⇒b phải có độ dài chẵn, nếu không thì họ không thể gặp nhau.
  Ví dụ: Nếu khoảng cách là 3 thì họ sẽ không thể dừng chung tại 1 ga
- Độ dài đường đi từ a đến điểm gặp nhau, và b đến điểm gặp nhau phải nhỏ hơn độ dài của dax và dyb (dab/2 <= dax && dab/2 <= dyb)

TH2: Gặp nhau khi lướt qua nhau 1 lượt
⇒ Lúc này dab đã tăng thêm n (do đi thêm 1 vòng)
⇒ dabn = dab+n

- Độ dài dabn phải có độ dài chẵn (bởi vì giống TH1)
- Độ dài đường đi từ A,B⇒ điểm gặp nhau phải nhỏ hơn độ dài của dax và dyb (dabn/2 <= dax && dabn/2 <= dyb)

Bài 2: Đổi nhị phân

Số 101 ở dạng nhị phân là

1	1	0	0	1	0	1

Tức là 1*2^0 + 0*2^1 + 1*2^2 + 0*2^3 + 0*2^4 + 1*2^5 + 1*2^6

= 2^6+2^5+2^2+2^0=64+32+4+1

Ở bài này chúng ta không nên dùng những hàm hay cấu trúc dữ liệu đặc biệt, để luyện tập sự khéo léo trong việc sử dụng vòng lặp

Ta gọi x là số lũy thừa của 2 lớn nhất mà nhỏ hơn n

⇒ Như ở bài trên với n = 101 thì x là 64 ⇒ chứa bit 1 ở vị trí số 7 từ phải sang

sau khi lấy n - x thì ta cần tìm tiếp biểu diễn nhị phân của 6 số tiếp theo, ta dùng vòng lặp để xử lý đoạn này

Bài 3: Chia quà (CUT)

Giả sử số lần cắt theo chiều ngang là a, số lần cắt theo chiều dọc là b

Thì số bánh cắt được là (a+1)*(b+1)

Dễ thấy, muốn tích này có kết quả lớn nhất ⇔ a và b gần bằng nhau nhất

Bài 4: Số fibonacci

Ở bài này, ta sử dụng 3 biến a,b và c lần lượt biểu diễn số cách 2 bước, số liền trước và số hiện tại

Ở mỗi vòng lặp ta gán
c = a+b; // Tính tổng số hiện tại = 2 số liền trước
a = b; // Đặt a = số phía trước của số hiện tại
b = c; // Đặt b = số hiện tại
Ta cần đặt như vậy để chuẩn bị cho vòng lặp tiếp theo (khi đó a cần là số liền trước, b là số cách 2 bước về trước và c lúc đó sẽ tính là số hiện tại)
*Lưu ý ở bài này ta sử dụng unsigned long long, vì số 92 đã tiệm cần gần 1e18 ⇒ số thứ 93 sẽ tràn só ⇒ dùng unsigned long long (kiểu dữ liệu lưu trữ các số trong khoảng [0,2e18]

Bài 5: Tìm số DNUMMAX

Ta đơn giản sẽ thử mọi số có các chữ số giống nhau (có 18*9 số như vậy)
Rồi ta tìm số lớn nhất mà nhỏ hơn số x

14/7/2025

Bài 1: Giả thuyết của GOLDBACH

=> các bạn chỉ cần duyệt tuần tự cho đến khi tìm được 2 số a,b thỏa mãn
Vì a+b=n và a<=b và a cần lớn nhất có thể. Do đó mình sẽ bắt đầu for a từ n/2 => 1 để tìm số thỏa mãn. Khi được cặp số a,b thì break trực tiếp
Code mẫu hàm check số nguyên tố

```cpp
bool isPrime(long long n){
    if(n<2) return false;
    for(int i =2; i *i<= n; i++){
        if(n%i == 0) return false;
    }
    return true;
}
```

Bài 2: Số bạn bè

=> Tính các ước thực sự của a và b, và so sánh.
Việc tìm các ước thực sự thì thuật toán các bạn chạy chỉ nên trong khoảng o(sqrt(n)). Bởi nếu chạy trong o(n) thì sẽ bị TLE

Bài 3: Số nguyên tố gần nhất

=> Đi về 2 phía từ a, nếu gặp số nguyên tố thì break. Trong trường hợp đồng thời gặp 2 số nguyên tố ở 2 phía thì sẽ chọn đứa giá trị nhỏ hơn

Bài 4: Cắt giấy

=> Nhận thấy nếu muốn chia mảnh giấy m*n cho k người thì tức là m*n%k==0
Do vậy bài toán trở về việc tìm số ước của m*n (Tuy nhiên vì m,n<=1e12) Nên việc lấy m*n rồi tìm ước như ở những bài trên sẽ bất khả thi
=> Thay vào đó, ta thấy rằng m và n có thể được biểu diễn qua tích của các số nguyên tố
m = (2^m1)*(3^m2)*(5^m3)*.....
n = (2^n1)*(3^n2)*(5^n3)*....
Với m1,m2,m3….; n1,n2,n3,.... >= 0

=> m*n = (2^(m1+n1))*(3^(m2+n2))*....
Gọi g là ước của m*n => g = (2^g1)*(3^g2)*(5^g3)*...
Với g1 <= m1+n1; g2 <=m2+n2; …..
Do vậy ta thấy số ước của m*n chính là tổ hợp cách ghép chọn của g1,g2,g3
⇒ Số ước của m*n là (m1+n1+1)*(m2+n2+1)*.....