KĨ THUẬT GHI NHỚ, CỘNG DỒN

Giáo viên: Nguyễn Như Thắng- THPT Chuyên Lào Cai

Việc lưu trữ thông tin cần nhớ sao cho có thể lấy lại chúng một cách nhanh nhất là một trong các kỹ năng cơ bản đầu tiên để có thể có một chương trình hiệu quả. Tuy vậy việc nhớ cái gì và lưu trữ như thế nào lại đòi hỏi sự nhạy bén toán học của học sinh. Điều này lại chỉ được hình thành sau khi học sinh tiếp xúc với một hệ thống các bài toán được tổ chức cẩn thận. Hệ thống này giúp học sinh xây dựng được các thói quen tư duy cơ bản cũng như các kỹ thuật cơ bản trong lập trình. Dưới đây, tôi trình bày một hệ thống các bài tập được phân loại nhằm mục đích hình thành cho các em kỹ năng nói trên.

Ưu điểm của kĩ thuật đánh dấu, ghi nhớ: lấy thông tin đã lưu trữ trong thời gian O(1).

1. Một số bài tập hình thành kĩ năng

1.1. Bài toán 1:

Cho mảng n số nguyên dương $a_1, a_2, \ldots, a_n$ $(1 \le n \le 10^6, a_i \le 10^6)$ và số nguyên dương S $(S \le 10^6)$. Hãy đếm xem có bao nhiêu cặp (a_i, a_j) thỏa mãn $a_i + a_j = S$.

Ví dụ:

Input	Output
5 2 1 1 1 1 1	10

a) Hướng dẫn thuật toán

- Thuật toán 1: Do biểu thức cần thỏa mãn $a_i + a_j = S$ có chứa 2 chỉ số là i, j do đó cách đơn giản nhất mà học sinh có thể nghĩ tới đó là dùng 2 vòng lặp để đếm xem có bao nhiêu cặp có tổng bằng S. Độ phức tạp là $O(n^2)$.

```
for(int i = 1; i < n; i++)
    for(int j = i + 1; j <= n; j++)
      if (a[i] + a[j] == s)
        res++; //ket qua
```

- Thuật toán 2: Ta có $a_i + a_j = S$ ➔ $a_i = S - a_j$. Do đó tại vị trí j đang xét, ta xem có bao nhiêu số a_i đã xuất hiện mà bằng $S - a_j$. Khi đó độ phức tạp của thuật toán là $O(n)$.

```
for(int i = 1; i <= n; i++) {
    cin >> a[i]; //doc du lieu
    res = res + cnt[s - a[i]]; //tang ket qua
    cnt[a[i]]++; //dem so lan a[i] xuat hien
  }
```

Đây cũng có thể gọi là kĩ thuật phân li ẩn số, ban đầu 2 chỉ số nằm cũng một vế của đẳng thức $a_i + a_j = S$. Sau đó thì ta phân li ẩn số như sau:

$a_i = S - a_j$. Mỗi khi đọc số a_j vào thì ta đánh dấu xem nó xuất hiện bao nhiêu lần, với mỗi a_j thì ta đếm xem có bao nhiêu số đã bằng $S - a_j$ đã xuất hiện.

b) Chương trình đầy đủ

```
#include<bits/stdc++.h>
using namespace std;
long long res, a[10001], n, s, cnt[10001];
int main() {
  cin >> n >> s;
  cnt[0] = 1;
  for(int i = 1; i <= n; i++) {
    cin >> a[i]; //doc du lieu
    res = res + cnt[s - a[i]]; //tang ket qua
    cnt[a[i]]++; //dem so lan a[i] xuat hien
  }
  cout << res;
}
```

Kĩ thuật ghi nhớ thông thường sẽ luôn đi cặp với việc cộng dồn qua bài toán sau.

1.2. Bài toán 2:

Cho một dãy số gồm n ($1 \le n \le 10^6$) số nguyên dương $a_i \le 10^6$, hãy xác định xem tồn tại bao nhiêu dãy con liên tiếp có tổng bằng S ?

Ví dụ:

Input	**Output**
5 3 1 1 1 1 1	3

a) Hướng dẫn thuật toán

- Thuật toán 1: Theo cách đơn giản nhất ta sẽ tính tổng tất cả các đoạn có thể $[i..j]$ nếu đoạn nào có tổng bằng S thì tăng biến res. Độ phức tạp $O(n^3)$.

```
for (int i = 1; i <= n; i++)
    for (int j = i; j <= n; j++) {
      long long tong = 0;
      for (int k = i; k <= j; k++) //Tinh tong doan [i..j]
        tong = tong + a[k];
      if (tong == s)
        res++; //ket qua
    }
```

- Thuật toán 2: Nhận xét thấy việc tính tổng được lặp đi, lặp lại. Tính tổng các đoạn [i..j] không có tính kế thừa. Do đó việc tính tổng mất thời gian $O(n)$. Để giảm thời gian tính tổng từ $O(n)$ về $O(1)$, ta cần chuẩn bị trước một mảng tổng cộng dồn sum[i] là tổng các số từ 1→i. Khi đó: `tong=sum[j]-sum[i-1].`

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

```
        if (sum[j] - sum[i - 1] == s)
          res++;
      }
```

- Thuật toán 3: Do dãy số là nguyên dương nên mảng tổng cộng dồn tăng liên tục. Vậy với mỗi vị trí j ta cần tìm các vị trí i sao cho `sum[j]-sum[i-1]=s` bằng chặt nhị phân. Độ phức tạp $O(nlogn)$.

```
for(int i = 1; i <= n + 1; i++)
res = res +
      (upper_bound(sum,sum+1+i,sum[i]-s)
       - lower_bound(sum,sum+1+i,sum[i]-s));
```

- Thuật toán 4: Sử dụng kĩ thuật ghi nhớ. Tương tự bài 1

b) Chương trình đầy đủ

```
#include<bits/stdc++.h>
using namespace std;
long long res, a[10001], n, s, cnt[10001], sum[10001];
int main() {
  cin >> n >> s;
  cnt[0] = 1;
  for(int i = 1; i <= n; i++) {
    cin >> a[i]; //doc du lieu
    sum[i] = sum[i - 1] + a[i];
    res = res + cnt[sum[i] - s];
    cnt[sum[i]]++;
  }
  cout << res;
}
```

1.3. Bài toán 3: Cho một dãy số gồm n $(1 \leq n \leq 10^6)$ số nguyên $|a_i| \leq 10^9$, hãy tìm giá trị lớn nhất của tổng dãy con liên tiếp của dãy ban đầu?

Ví dụ:

Input	**Output**
5 13 -1 -10 15 1	18

a) Hướng dẫn thuật toán

Đây là một trong những bài toán điển hình cho việc cải tiến chương trình sao cho mức độ hiệu quả ngày càng cao hơn.

- Thuật toán 1: Ta xét thuật toán đơn giản nhất để giải bài toán trên có độ phức tạp $O(n^3)$ như sua: Xét tất cả các đoạn $[i,j]$ với mỗi đoạn đó lại tính tổng các phần tử của nó, nếu tổng này lớn hơn giá trị lớn nhất đang có thì cập nhật lại giá trị lớn nhất.

```
for (int i = 1; i <= n; i++)
    for(int j = i; j <= n; j++) {
      int tong = 0;
      for(int k = i; k <= j; k++)
```

```
        tong = tong + a[k];
      if (tong > res)
        res = tong;
    }
```

- Thuật toán 2: Tương tự bài toán 2, ta sẽ cộng dồn trước. Độ phức tạp $O(n^2)$.

```
for(int i = 1; i <= n; i++) {
    cin >> a[i];
    s[i] = s[i - 1] + a[i]; //cong don
  }
  res = a[1];
  for (int i = 1; i <= n; i++)
    for(int j = i; j <= n; j++) {
      if (s[j] - s[i - 1] > res)
        res = s[j] - s[i - 1]; //cap nhat kq
    }
```

- Thuật toán 3: Nhận xét thấy để $\{s[j] - s[i-1]\}_{\max}$ thì với $s[j]$ cố định thì ta phải đi tìm $s[i-1]_{\min}$. Do đó ta duy trì tìm $s_{\min}$ cùng với tìm res.

Độ phức tạp thuật toán trên là $O(n)$.

b) Chương trình đầy đủ

```
#include<bits/stdc++.h>
using namespace std;
long long n, a[100005], res, s[100005], smin;
int main() {
  cin >> n;
  for(int i = 1; i <= n; i++) {
    cin >> a[i];
    s[i] = s[i - 1] + a[i]; //cong don
  }
  smin = s[0], res = a[1]; //khoi tao smin, res
  for(int i = 1; i <= n; i++) {
    if (s[i] - smin > res)
      res = s[i] - smin; //cap nhat lai kq
    if (s[i] < smin)
      smin = s[i]; //cap nhat lai smin
  }
  cout << res;
}
```

1.4. Bài toán 4: Bộ ba số Pitago

Cho dãy số nguyên dương gồm $\boldsymbol{n}$ phần tử $\boldsymbol{a_1, a_2, \ldots a_n}$ $\boldsymbol{(0 < a_i \leq 10^4, n \leq 10^4)}$,

Hỏi có bao nhiêu bộ 3 số $\boldsymbol{a_i, a_j, a_k}$ thỏa mãn $\boldsymbol{a_i^2 + a_j^2 = a_k^2}$ (*với i, j, k đôi một khác nhau)* bộ 3 số này được gọi là bộ ba số Pitago.

Dữ liệu vào: Dòng đầu là số $\boldsymbol{n}$; dòng tiếp theo là $\boldsymbol{n}$ số của dãy đã cho.

Kết quả ra: Ghi ra số lượng bộ 3 số Pitago, nếu không có thì ghi ra 0.

Ví dụ:

PITAGO.INP	PITAGO.OUT	Giải thích
5 5 12 4 3 13	2	Có 2 bộ 3 số Pitago là: (3, 4, 5) và (5, 12, 13).
3 4 5 6	0	

Ràng buộc: 30% test có $n \leq 100$;

30% test có $100 < n \leq 1000$;

40% test có $1000 < n \leq 10000$

a) Hướng dẫn thuật toán

Nhận xét: Với bộ ba số (a_i, a_j, a_k) thỏa mãn $a_i^2 + a_j^2 = a_k^2$ thì chắc chắn đã lập thành số đo 3 cạnh của 1 tam giác, vì:

$$(a_i - a_j)^2 \leq a_i^2 + a_j^2 \leq (a_i + a_j)^2$$

$$\rightarrow |a_i - a_j| \leq a_k \leq a_i + a_j$$

- Thuật toán 1:Duyệt toàn bộ các khả năng của bộ 3 số a_i, a_j, a_k bằng 3 vòng lặp lồng nhau. Nếu thỏa mãn $a_i^2 + a_j^2 = a_k^2$ thì tăng biến đếm lên 1.

```
for(int i=1;i<=n-2;i++)
   for (int j=i+1;j<=n-1;j++)
     for(int k=j+1;k<=n;k++)
        if (a[i]*a[i]+a[j]*a[j]==a[k]*a[k]) res++;
```

Độ phức tạp $O(n^3)$.

- Thuật toán 2: Ta xét bình phương các độ dài, giả sử các giá trị đố được sắp xếp tăng thì $a_k \geq a_j \geq a_i$ và $k \geq j \geq i$. Có thể tìm k bằng tìm kiếm nhị phân sử dụng hàm có sẵn lower_bound và upper_bound như sau:

```
for(int i=1;i<=n-2;i++)
   for (int j=i+1;j<=n-1;j++)
      res=res+upper_bound(a+1+j,a+1+n)- lower_bound(a+1+j,a+1+n);
```

Độ phức tạp sẽ là $O(n^2 logn)$.

- Thuật toán 3: Sử dụng mảng đánh dấu $d[100000005]$, đánh dấu và đếm xem a_k^2 xuất hiện bao nhiêu lần.

Duyệt qua tất cả các cặp $a_i^2 + a_j^2$, cộng $d[a_i^2 + a_j^2]$ vào kết quả.

Độ phức tạp $O(n^2)$.

b) Chương trình đầy đủ:

```
#include<bits/stdc++.h>
#define FOR(i,a,b) for(int i=a;i<=b;i++)
using namespace std;
int n,a[100005],d[100000005],res;
int main() {
    freopen("PITAGO.inp","r",stdin);
    cin>>n;
    FOR(i,1,n) {
```

```
        cin>>a[i];
        a[i]=a[i]*a[i]; //luu binh phuong do dai
        d[a[i]]++; //dem so lan xuat hien
    }
    sort(a+1,a+1+n); ///sap xep
    FOR(i,1,n-2)
    FOR(j,i+1,n-1) {
        res+=d[a[i]+a[j]]; //tinh ket qua
    }
    cout<<res;
}
```

1.5. Bài toán 5: Dance

Lớp học múa khiêu vũ dạ hội của giáo sư Padegras có ***n*** học sinh nam và nữ ghi tên. Giáo sư cho tất cả học sinh xếp thành một hàng dọc và chọn một nhóm các học sinh liên tiếp nhau cho buổi học đầu tiên với yêu cầu là số học sinh nam và nữ phải bằng nhau.

Hãy xác định, giáo sư Padegras có bao nhiêu cách lựa chọn khác nhau cho buổi học đầu tiên.

Input:

- Dòng đầu tiên chứa số nguyên ***n*** ($1 \le n \le 10^6$),
- Dòng thứ 2 chứa xâu độ dài ***n*** bao gồm các ký tự từ tập {***a***, ***b***} xác định dòng xếp hàng, ***a*** là nam, ***b*** – nữ.

Output: Một số nguyên – số cách lựa chọn.

Example:

Input:	**Output:**
8 abbababa	13

Ràng buộc: 30% test có $n \le 200$
40% test có $200 < n \le 3000$
30% test có $3000 < n \le 1000.000$

a) Hướng dẫn thuật toán:

- *Thuật toán 1:* Xét mỗi đoạn [i..j], đếm số bạn nam, đếm số bạn nữ, nếu bằng nhau thì tăng biến đếm lên 1. Độ phức tạp $O(n^3)$. Chỉ được 30% test.

- *Thuật toán 2:* Tư tưởng cũng cơ bản giống thuật toán 1, tuy nhiên sử dụng 2 mảng cộng dồn sa[] và sb[] để tính số bạn nam và bạn nữ, từ đó có thể trả lời xem đoạn [i..j] có số bạn nam bằng số bạn nữ không trong O(1) vì chỉ cần so sánh $sa[j] - sa[i-1] == sb[j] - sb[i-1]$.

Độ phức tạp: $O(n^2)$. Chỉ được thêm 40% test.

- *Thuật toán 3:* Từ nhận xét của thuật toán 2, $sa[j] - sa[i-1] == sb[j] - sb[j-1]$. Ta có:

$sa[j] - sb[j] == sa[i-1] - sb[i-1]$, do đó có thể đếm được ngay tất cả các đoạn thỏa mãn mà kết thúc tại j bằng cách đếm xem hiệu $sa[j] - sb[j]$ đã xuất hiện bao nhiêu lần trước đó. Đây là kĩ thuật đánh dấu.

Chú ý $sa[j] - sb[j]$ có thể âm, do đó ta cần xử lý bằng cách cộng hiệu này với n để đảm bảo rằng không sử dụng mảng có chỉ số âm.

Độ phức tạp: O(n).

Ta quy ước nam là 0 còn nữ là 1.

b) Chương trình đầy đủ

```
#include<bits/stdc++.h>
using namespace std;
int n, a[1000005], sa[1000005], sb[1000005], d[2000005];
int main() {
  for(int i = 1; i <= n; i++) {
    char c;
    scanf("%c", &c);
    if (c == 'a')
      a[i] = 0;
    else
      a[i] = 1;
  }
  sa[0] = 0;
  for(int i = 1; i <= n; i++)
    if (a[i])
      sa[i] = sa[i - 1] + 1;
    else
      sa[i] = sa[i - 1];
  sb[0] = 0;
  for(int i = 1; i <= n; i++)
    if (!a[i])
      sb[i] = sb[i - 1] + 1;
    else
      sb[i] = sb[i - 1];
  for(int i = 0; i <= 2 * n; i++)
    d[i] = 0;
  d[n] = 1;
  long long res = 0;
  for(int i = 1; i <= n; i++) {
    int t = n + (sa[i] - sb[i]);
    //dam bao sa[i]-sb[i] luon duong
    res += d[t];
    d[t]++;
  }
  printf("%I64d", res);
}
```

1.6. Bài toán 6: Phần thưởng

Tuấn là người chiến thắng trong một cuộc thi "tìm hiểu kiến thức vũ trụ" và được nhận các phần thưởng do công ty XYZ tài trợ. Các phần thưởng được bố trí trên một bảng hình vuông kích thước $\boldsymbol{n} * \boldsymbol{n}$ có dạng một lưới ô vuông kích thước đơn vị. Các dòng của bảng được đánh số từ **1** đến $\boldsymbol{n}$, từ trên xuống dưới và các cột của bảng được đánh số từ **1** đến $\boldsymbol{n}$, từ trái qua phải. Ô nằm trên giao của dòng **i** và cột **j** được gọi là ô $(\boldsymbol{i}, \boldsymbol{j})$ và trên ô đó chứa một món quà có giá trị là $\boldsymbol{a}[\boldsymbol{i}, \boldsymbol{j}]$ $(\mathbf{1} \leq \boldsymbol{i}, \boldsymbol{j} \leq \boldsymbol{n})$.

Để nhận phần thưởng, Tuấn được chọn một hình vuông $\boldsymbol{k} * \boldsymbol{k}$ chiếm trọn trong một số ô của bảng và nhận tất cả các phần quà có trong các ô nằm trong đó.

Yêu cầu: Hãy xác định tổng giá trị quà lớn nhất mà Tuấn có thể nhận được.

Dữ liệu vào: Dòng thứ nhất chứa hai số nguyên dương **n, k** $(n \leq 10^3, \frac{n}{3} \leq k \leq n)$. Dòng thứ i trong số n dòng tiếp theo chứa n số nguyên dương, số thứ j là a[i,j] (a[i,j] ≤ 1000)

Kết quả ra: Ghi ra một số nguyên duy nhất là tổng giá trị lớn nhất của các món quà mà Tuấn có thể nhận được.

Ví dụ:

BONUS.INP	BONUS.OUT	Giải thích
4 3 1 9 1 1 9 9 9 9 1 9 9 9 1 9 9 14	86	Chọn vùng 3x3 như sau: 1 9 1 1 9 **9 9 9** 1 **9 9 9** 1 **9 9 14**

Ràng buộc: *50% test có* $\boldsymbol{n} \leq \mathbf{100}$

50% test có $\mathbf{100} < \boldsymbol{n} \leq \mathbf{1000}$

a) Hướng dẫn thuật toán: Độ phức tạp là $O(n^2)$.

- Thuật toán 1: Dễ thấy thuật toán đơn giản như sau:

Xét tất cả các hình vuông $k * k$ có đỉnh góc trái trên là (i, j). Độ phức tạp là $O(n^2 * k^2)$.

- Thuật toán 2: Tương tự các bài trước, ta thấy việc tính tổng các số trong hình vuông $k * k$ không có tính kế thừa với thuật toán 1. Để cải thiện việc này ta tạo mảng $s[i][j]$ có ý nghĩa là tổng các ô trong hình vuông có góc trái trên là (1,1) và góc trái dưới là (i,j). Khi đó hình vuông k*k có góc trái trên là ô (i,j) được tính như sau:

Tong(i,j)=s(i+k-1,j+k-1)-s(i-1,j+k-1)-s(i+k-1,j-1)+s(i-1,j-1).

Với **s[u,v]=s[u-1,v]+s[u,v-1]-s[u-1,v-1]+a[u,v]**

b) Chương trình đầy đủ:

```
#include<bits/stdc++.h>
using namespace std;
int n, k, a[1001][1001], s[1001][1001], res;
int main() {
```

```
  cin >> n >> k;
  for(int i = 1; i <= n; i++)
    for(int j = 1; j <= n; j++) {
      cin >> a[i][j];
      s[i][j] = s[i-1][j]+s[i][j-1]-s[i - 1][j - 1] + a[i][j];
    }
  for(int i = 1; i <= n - k + 1; i++)
    for(int j = 1; j <= n - k + 1; j++) {
      int tong = s[i + k - 1][j + k - 1] - s[i - 1][j + k - 1]
                 - s[i + k - 1][j - 1] + s[i - 1][j - 1];
      if (tong > res)
        res = tong;
    }
  cout << res;
}
```

1.7. Bài toán 7. Phần thưởng 2

Mở rộng cho bài toán 6, khi đó không biết k, các số có thể âm và $n \leq 400$.

Ví dụ:

BONUS2.INP	BONUS2.OUT	Giải thích
4 3 1 -9 1 1 -9 9 9 9 1 9 9 9 1 9 9 14	86	Chọn vùng 3x3 như sau: 1 -9 1 1 -9 **9 9 9** 1 **9 9 9** 1 **9 9 14**

a) Hướng dẫn giải thuật: Dễ dàng thấy là có thể dựa hoàn toàn theo bài 6. Độ phức tạp thuật toán là $O(n^3)$.

```
for(int k = 1; k <= n; k++)
    for(int i = 1; i <= n - k + 1; i++)
      for(int j = 1; j <= n - k + 1; j++) {
        int tong = s[i + k - 1][j + k - 1] - s[i - 1][j + k - 1]
                   - s[i + k - 1][j - 1] + s[i - 1][j - 1];
        if (tong > res)
          res = tong;
      }
```

1.8. Bài toán 8. Phần thưởng 3

Mở rộng bài toán 7, cho hình chữ nhật $n * m$ chứa số $|a_{ij}| \leq 10^6$ cần chọn ra hình chữ nhật có các cạnh song song với hình chữ nhật ban đầu sao cho có tổng là lớn nhất.

Ví dụ:

BONUS3.INP	BONUS3.OUT	Giải thích
3 4 -1 1 -1 -1 -2 -2 9 -1	13	Chọn từ (1;1) đến (3;3)

9 0 -1 -1		

a) Hướng dẫn giải thuật: Độ phức tạp là $O(n^2 * m^2)$ ngay cả khi áp dụng cách làm của bài 6.

Để giảm độ phức tạp đa thức giảm còn bậc 3, ta sử dụng kĩ thuật đánh dấu, cộng dồn, tính theo từng khe. Kết hợp cách làm của bài toán 3. Độ phức tạp $O(n^2 * m)$.

b) Chương trình đầy đủ:

```
#include <iostream>
using namespace std;
long long s[401][401],a[401][401],res;
int n,m;
long long get(int i,int j, int u,int v) { //lay tong mot HCN
  return (s[u][v]-s[i-1][v]-s[u][j-1]+s[i-1][j-1]);
}
int main() {
  cin>>n>>m;
  for(int i=1; i<=n; i++)
    for(int j=1; j<=m; j++) {
      cin>>a[i][j];  //doc du lieu
    }
  for(int i=1; i<=n; i++)
    for(int j=1; j<=m; j++) {
      //tong cong don
      s[i][j]=s[i-1][j]+s[i][j-1]-s[i-1][j-1]+a[i][j];
    }
  for(int i=1; i<=n; i++)
    for(int j=i; j<=n; j++) {
      long long smin=0, t[401]= {0};
      for(int k=1; k<=m; k++) { //tim min theo khe
        t[k]=t[k-1]+get(i,k,j,k);
        res=max(t[k]-smin,res); //cap nhat GTLN neu co
        smin=min(smin,t[k]); //duy tri smin
      }
    }
  cout<<res;
}
```

Một điều thú vị là kỹ thuật cố định hàng trên và hàng dưới là kỹ thuật phổ biến để đưa việc giải bài toán hai chiều về bài toán một chiều.

1.9. Bài toán 9. Chống đẩy

Nguồn: Đề thi đề xuất Duyên Hải 2017 lớp 10 của Hưng Yên.

Hai đội *Bình Minh* và *Rạng Đông* thi đấu. Mỗi khi một đội ghi điểm, các cầu thủ đội còn lại phải chống đẩy số lần đúng bằng số điểm hiện tại của đội đối phương. Ví dụ, lần đầu đội Bình Minh ghi 7 điểm, đội Rạng Đông chống đẩy 7 lần. Lần thứ hai ghi 3 điểm (được tổng 10 điểm), đội Rạng Đông phải chống đẩy

10 lần. Tương tự, đội Bình Minh ghi tiếp 2 điểm, đội Rạng Đông chống đẩy tiếp 12 lần. Tổng số lần chống đẩy trong trường hợp này là $7 + 10 + 12 = 29$.

An là một thành viên ở đội Rạng Đông. An đếm được mình đã chống đẩy tất cả N lần. Hỏi số điểm tối đa đội Bình Minh đã ghi được là bao nhiêu?

Dữ liệu: vào từ file **PUSHUPS.INP**

- Dòng đầu chứa hai số nguyên dương N, M ($N \leq 5000,\ M \leq 10$) với N - số lần An chống đẩy trong trận đấu, M – số cách ghi điểm mà đội Bình Minh có thể thực hiện.

- Dòng thứ hai chứa M số nguyên dương $S_i (S_i \leq 20)$ là số điểm tương ứng mà đội Bình Minh có thể ghi theo M cách tương ứng. Mỗi cách ghi điểm có thể thực hiện nhiều lần.

Kết quả: Ghi ra file **PUSHUPS.OUT** một số nguyên duy nhất là số điểm tối đa mà đội Bình Minh đã ghi được. Ghi ra -1 nếu không tìm ra cách ghi điểm thỏa mãn trong trường hợp An nhớ nhầm.

PUSHUPS.INP	PUSHUPS.OUT	GIẢI THÍCH
29 3 7 2 3	14	Số điểm lần lượt ghi là 3, 2, 2, 7. Số lần chống đẩy $3 + 5 + 7 + 14 = 29$.

a) Hướng dẫn giải thuật

Thuật toán 1: Xác định đây là dạng bài quy hoạch động. Bài toán cần xác định công thức truy hồi với 2 biến là số lần chống đẩy và số điểm ghi được.

Gọi công thức truy hồi là $dp[i][j]$ trong đó i là số lần chống đẩy, j là số điểm đạt được.

Với mỗi lần ghi được điểm s_k thì ta đánh dấu $dp[i + j + s_k][j + s_k]$

Đến cuối cùng nếu đánh dấu được $dp[n][x]$ nào đó thì tồn tại kết quả của bài toán, nếu không thì An nhớ nhầm.

Duyệt $dp[n][x]$ để tìm x lớn nhất có thể là xong.

Độ phức tạp thuật toán: $O(n^2 * m)$.

```
dp[0][0] = 1;
for (int i = 0; i < n; i++)
   for (int j = 0; j <= i; j++)
      if (dp [i][j])
         for (int k = 0; k < m; k++)
             if (i+j+s[k] <= n)
                 dp [i+j+s[k]][j+s[k]] = 1;
 for (int x = n; x >= 0; x--)
    if (dp [n][x]) {
       cout << x;
       return 0;
    }
 cout << -1;
```

Thuật toán 2: Nhận xét thấy số lần chống đẩy i tăng nhanh hơn nhiều so với số điểm ghi được j. Ta có thể điều chỉnh lại một chút thuật toán 2 để có độ phức tạp $O(n * \sqrt{n} * m)$.

b) Chương trình đầy đủ

```
#include <bits/stdc++.h>
#define maxn 5001
using namespace std;
int n, m, s[11];
bool dp[maxn][maxn];
int main() {
//  freopen("pushups.inp", "r", stdin);
//  freopen("pushups.out", "w", stdout);
  cin >> n >> m;
  for (int i = 0; i < m; i++)
    cin >> s[i];
  int M=*max_element(s,s+m);
  memset(dp, 0, sizeof(dp));
  dp[0][0] = 1;
  for (int i = 0; i < n; i++)
    for (int j = 0; j <= sqrt(i)+M; j++)
      if (dp[i][j])
        for (int k = 0; k < m; k++)
          if (i+j+s[k] <= n)
            dp[i+j+s[k]][j+s[k]] = 1;
  for (int x = n; x>=0; x--)
    if (dp[n][x]) {
      cout << x;
      return 0;
    }
  cout << -1;
}
```

2. Một số bài tập vận dụng

- Bài 1: http://www.spoj.com/THPTCBT/problems/YB_KT2B2/
- Bài 2: Bonus2 (Bài 4.52 – Sách bài tập chuyên tin quyển 1).
- Bài 3: Bonus (HSG quốc gia 2018) http://vn.spoj.com/problems/BONUS13/
- Bài 4: **Đổi tiền** – làm tương tự bài PUSHUPS

Cho một ngân hàng có N loại tiền mệnh giá A[1],A[2],...A[N] với số lượng tiền mỗi loại không giới hạn. Cần chi trả cho khách hàng một số tiền M đồng. Hãy cho biết cần bao nhiêu tiền mỗi loại để chi trả sao cho số lượng tờ là ít nhất .

Dữ liệu vào từ file: **tien.inp** như sau :

- Dòng đầu tiên ghi 2 số N, M. $(N \leq 100, M \leq 10^4)$.
- Dòng thứ hai ghi N Số: $A_1, A_2, A_3, \dots A_N$.

Kết quả: Ghi ra file **tien.out** như Sau :

- Dòng Đầu tiên ghi số tờ cần dùng. Nếu không thể đổi được thì ghi số 0 và không cần thực hiện tiếp.
- Dòng tiếp theo ghi số n số biểu hiện cho số tờ cần dùng cho mỗi loại.

- Bài 5: Link sau https://drive.google.com/file/d/1sYclamC9NUm-XZ4nOU0ZmMklTjCtd1P8/view?usp=sharing