Sol bài SZERO:

Ta cần đếm xem có bao nhiêu đoạn có tổng = 0 trong dãy đã cho.

Thuật trâu:

Dùng hai con trỏ i và j chạy lần lượt, tính tổng đoạn từ i đến j: nếu nó có tổng = 0 thì tăng biến đếm:

```
for(int i = 1; i <= n; i++)
        for(int j = i; j <= n; j++)
        {
            long long sum = 0;
            for(int k = i; k <=j; k++) sum += a[k];
            if(sum == 0)
                if(cnt < j - i + 1)cnt = j - i + 1;
        }
```

Cài thuật toán này bạn có 55 điểm.

Cải tiến: Ta có thể tính trước mảng tổng để với mỗi cặp i,j thay vì phải dùng thêm 1 vòng for để tính tổng ta sẽ tính bằng 1 phép tính:

```
sum[0] = 0;
    for(int i = 1; i <=n; i++)
    {
        cin >> a[i];
        sum[i] = sum[i-1]+ a[i];
    }
long long cnt = 0;
    for(int i = 1; i <= n; i++)
        for(int j = i; j <= n; j++)
        {
            long long s = 0;
            s = sum[j] - sum[i - 1];
            if(s == 0)
                if(cnt < j - i + 1)cnt = j - i + 1;
        }
```

Thuật chuẩn:

Thuật trên vẫn lâu vì phải mất 2 vòng for để xét tất cả các đoạn con của dãy. Ta còn 1 nhận xét nữa: Để 1 đoạn (i, j) có tổng = 0 thì sum[i-1] và sum[j] phải bằng nhau. Như vậy, trick ở đây là khi xét trên mảng sum đã tính được trước,với mỗi giá trị sum[i] ta chỉ xét những thằng bằng nó để tìm ra đoạn có tổng = 0 → cần sắp xếp sum để những giá trị bằng nhau sẽ đứng cạnh nhau → xét sẽ nhanh hơn.

Tuy nhiên, khi sort mảng sum ta vẫn không được làm "mất dấu" chỉ số i, j ban đầu của nó để còn tìm độ dài đoạn dài nhất có tổng = 0 → hoặc dùng bản ghi để ghi lại cả chỉ số lẫn giá trị của mảng, hoặc dùng sắp xếp kiểu chỉ số. Ta sẽ xét lần lượt 2 cách:

Cách 1: sắp xếp theo chỉ số:

Dùng thêm 1 mảng id[i] = i để ghi nhận chỉ số mảng ban đầu. Sau đó ta sẽ sắp xếp mảng id theo tiêu chí sum[i] < sum[j] thì i < j:

```
bool lf(int i, int j)
{
    return sum[i] < sum[j];
}

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

```
        id[i] = i;
    stable_sort(id, id + n + 1, lf);
}
```

Hàm stable_sort khác hàm sort ở chỗ là nếu sum[i] == sum[j] thì nó chỉ số nào đứng trước nó vẫn đứng trước, chỉ số nào đứng sau nó vẫn ở sau.

Ví dụ, ta có mảng:

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

mảng sum:

0 2 9 14 11 9 13 4 2 3

và mảng id sau khi sắp xếp:

0 1 8 9 7 2 5 4 6 3

Rõ ràng, khi đọc giá trị mảng sum theo dãy id đã sắp, ta có mảng sum được sắp xếp tăng dần.

Giờ xét từng đoạn trên mảng sum có giá trị bằng nhau (dùng kĩ thuật hai con trỏ):

i = 0, j = i + 1(chạy sau i)

Chừng nào sum[j]== sum[i] thì j++

- ➔ vòng lặp này dừng ở vị trí j đầu tiên mà sum[j] != sum[i] → đoạn từ sum[i] đến sum[j-1] gồm những giá trị bằng nhau.
- ➔ độ dài đoạn dài nhất có tổng = 0 là j – i

Vì các giá trị từ i ..j-1 là các giá trị bằng nhau rồi nên lần lặp tiếp theo cho i = j (vị trí j đầu tiên mà sum[j] != sum[i])

Tuy nhiên, áp dụng vào chương trình này ta phải thay đổi một chút vì ta xét trên mảng id

Ta có đoạn chương trình viết như sau:

```
int res = 0;
    int i = 0;
    do
    {
        int j = i + 1;
        while (j <= n && sum[id[i]] == sum[id[j]]) j++;
        int len = id[j - 1] - id[i];
        if (len > res) res = len;
        i = j;
    }
    while (i <= n);
    return res;
```

Cách 2: Dùng bản ghi:

Tạo 1 bản ghi gồm 2 trường: s: lưu tổng và id:lưu vị trí ban đầu của s trong dãy sum:

```
struct X{
  long long s;
  int id;
};
X sum[maxN];
```

Tính tổng:

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

```
    }
Sắp xếp:
bool lf(X i, X j)
{
    return i.s < j.s;
}

stable_sort(sum, sum + n + 1, lf);

Và xử:

int res = 0;
    int i = 0;
    do
    {
        int j = i + 1;
        while (j <= n && sum[i].s == sum[j].s) j++;
        int len = sum[j-1].id - sum[i].id;
        if (len > res) res = len;
        i = j;
    }
    while (i <= n);
    return res;
```

So sánh: nếu dùng bản ghi thì ta đã lưu thông tin về chỉ số cũ trên mỗi bản ghi rồi nên khi sort ta cứ trộn mảng sum theo giá trị s mà không cần phải quan tâm đến sự xáo trộn của nó nữa. Khi cần biết vị trí cũ ta truy cập phần tử sum[i].id là biết vị trí cũ của nó ở đâu trước khi sắp xếp.
Bạn dùng cách nào cũng được, miễn là hiểu và biết áp dụng cách làm này vào các bài khác.

Sol bài PS:

Bài này cần đưa ra một đoạn dài nhất gồm các số có tổng dương nên việc nhận xét tổng bằng nhau không còn đúng nữa, cũng không sắp xếp được mảng tổng do giá trị nhỏ hơn có thể lại đứng sau giá trị lớn hơn → không áp dụng được thuật toán bài SZERO

Thuật trâu thì thôi không mô tả nữa nhé.

Thuật chuẩn:

Bước 1: Tính mảng tổng cộng dồn sum.

Bước 2: Cần tính toán để trả lời câu hỏi: với sum[j] giá trị i trước j có sum[i] <sum[j] là giá trị nào?Liệu có tính trước được hay không?

Câu trả lời là được:

Gọi minid[i] là chỉ số của phần tử đạt min trong các giá trị sum[0]…sum[i]. Ta có nhận xét: minid[i] = i (nếu sum[i] là giá trị nhỏ hơn tất cả các sum từ 0 đến i-1)

nếu không (tức là trước i có giá trị sum[.] nhỏ hơn sum[i]) thì:

minid[i] = minid[i - 1] (minid[i- 1] chỉ số của phần tử đạt min trong các giá trị sum[0]…sum[i-1])

```
minid[0] = 0; s[i] = s[i - 1] + a;
for (int i = 1; i <= n; ++i)
    {
        cin >> a;
        s[i] = s[i - 1] + a;
        minid[i] = s[i] < s[minid[i - 1]] ? i : minid[i - 1];
    }
```

Xét ví dụ đề bài: Mảng sum, và minid:

i	0	1	2	3	4	5	6	7	8	9	10
a		-5	-2	-3	4	-6	7	-8	9	-1	-20
sum	0	-5	-7	-10	-6	-12	-5	-13	-4	-5	-25
minid	0	1	2	3	3	5	5	7	7	7	10

Chú ý: dãy chỉ số đạt các vị trí min là 0, 1, 2, 3, 5, 7, 10 (**dãy sum theo các chỉ số này là dãy giảm dần**)

Như vậy, nếu thử với a[10], ta biết rằng tổng cộng dồn của các phần tử từ 1 đến 10 =-25 và đây là giá trị nhỏ nhất từ đầu dãy, vậy minid[10]= 10, phần tử nhỏ nhất từ 0..9 là sum[7] nên minid[9]= 7….

Vậy khi tính được mảng minid, ta sẽ dựa trên nó để tìm đoạn dài nhất như sau:

Từ phần tử a[j], i = minid[j], nếu i != j (i < j) thì i + 1..j là 1 đoạn có tổng dương (vì chắc chắn sum[i] < sum[j]). Lại xét tiếp i = minid[i – 1] để liệu tiến ngược tiếp về bên trái liệu có thể tiếp tục tìm được đoạn có tổng vẫn dương không (nếu không thì phần tử j chỉ có thể "kéo dài" đến i vì càng ngược về trước xét theo mảng minid thì giá trị sum càng tăng – theo cái nhận xét được bôi vàng kia kìa). Vậy, ta lại sử dụng kĩ thuật hai con trỏ để tìm đoạn dài nhất theo cách mô tả như trên:

```
    int resi = 0, resj = 0;
    int i = minid[n], j = n;
    while (true)
    {
        while (j > i && s[j] <= s[i]) --j;
        if (j - i > resj - resi)
        {
            resi = i; resj = j;
        }
        if (i == 0) break;
        i = minid[i - 1];
    }
Đáp số bài toán sẽ là resi + 1…resj
```