DỊCH CHUYỂN VÒNG

Giáo sư X đưa các bé trường mầm non SuperKids thăm hồ Big-O, nơi có huyền thoại về sự xuất hiện của những người ngoài hành tinh. Các bé được vui chơi tự do và đến cuối ngày sẽ có các xe đón về.

Con đường bao quanh hồ Big-O có độ dài n km, dọc theo con đường có n bến xe cách đều nhau đánh số từ 1 tới n theo một chiều đi quanh hồ gọi là **chiều đánh số**. Có a_i bé đứng tại bến i.

Trường có k xe, mỗi xe sẽ được điều đến một bến nào đó để đợi đón các bé. Nếu một bé đứng ở bến không có xe đón, bé sẽ phải **di chuyển trên con đường theo chiều đánh số** cho tới bến có xe đón.

Giáo sư X muốn tìm vị trí các bến cho xe đợi ở đó sao cho tổng độ dài quãng đường các bé phải di chuyển là nhỏ nhất. Sau một hồi phân tích, ông nhận ra đó là một thách thức nổi tiếng của những người ngoài hành tinh để lại trên Trái Đất: Bài toán Circular-Shift Problem (CSP)

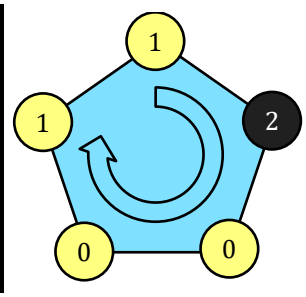
Dữ liệu: Vào từ file văn bản CSP.INP

- Dòng 1 chứa hai số nguyên dương n, k ($n \leq 800; k \leq n$) tương ứng là số bến và số lượng xe đón học sinh.
- Dòng 2 chứa n số nguyên $a_1, a_2, \ldots, a_n$ là số học sinh tại các bến. Tổng số học sinh không vượt quá 10^6.

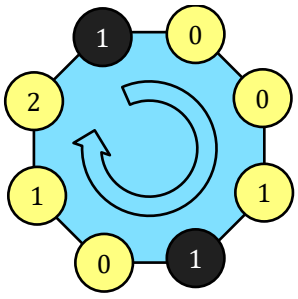
Các số trên một dòng của input file được ghi cách nhau bởi dấu cách

Kết quả: Ghi ra file văn bản CSP.OUT một số nguyên duy nhất là tổng độ dài quãng đường các bé phải di chuyển (tính bằng km) theo phương án tối ưu tìm được.

CSP.INP	CSP.OUT
5 1 1 2 0 0 1	3



CSP.INP	CSP.OUT
8 2 1 0 0 1 1 0 1 2	5



Bộ test chia làm các subtasks:

Subtask 1 (20% số điểm): $k \leq 3$

Subtask 2 (20% số điểm): $n \leq 60$

Subtask 3 (30% số điểm): $n \leq 300$

Subtask 4 (30% số điểm): Không có ràng buộc bổ sung

Thuật toán

Subtask 1 cần một thuật toán adhoc, nó cho gợi ý về cách tính tổng quãng đường di chuyển ứng với một phương án phân phối xe. Việc giải quyết subtask 1 với kích thước n lớn còn gợi ý dùng phương pháp 2 con trỏ, kỹ thuật sẽ được nâng cấp thành Knuth optimization trong subtask cuối cùng.

Subtask 2 gợi ý rằng bài toán này có thể giải bằng quy hoạch động:

Để tiện cho việc xử lý vòng tròn, ta tăng số bến lên gấp đôi, bến $n + 1$ trùng với bến 1, bến $n + 2$ trùng với bến 2, ..., bến $2n$ trùng với bến n. Khi đó mảng A được nhân đôi kích thước:

$$a_1, a_2, \dots, a_n, a_{n+1}, a_{n+2}, \dots, a_{2n}$$

$(a_{n+1} = a_1, a_{n+2} = a_2, \dots, a_{2n} = a_n)$

Bây giờ một đoạn liên tiếp gồm đúng n bến tương đương với n bến vòng quanh hồ theo chiều đánh số, vì vậy ta chỉ cho phép các học sinh di chuyển từ bến chỉ số nhỏ hơn tới bến chỉ số lớn hơn.

Gọi $Cost(i, j)$ là chi phí dồn tất cả các học sinh ở các bến $\in i \dots j$ về bến j. Hàm $Cost(i, j)$ có thể tính được trong thời gian O(1) (*)

Để ngắn gọn, ta gọi tắt việc "đón đoạn $i \dots j$ bởi p xe" nghĩa là đặt p xe vào các bến trong phạm vi từ i tới j để đón tất cả các học sinh trong phạm vi đó, các xe được đánh số từ 1 tới p theo thứ tự bến và xe cuối cùng (xe thứ p) đón tại bến j.

Gọi $f(p, i, j)$ là chi phí nhỏ nhất (tính bằng tổng độ dài quãng đường di chuyển của các học sinh) nếu đón đoạn $i \dots j$ bởi p xe. Lưu ý là nếu số xe dư thừa thì một vài xe có thể đón ở cùng bến, trong trường hợp này chỉ có một xe số hiệu nhỏ nhất tại bến đón học sinh, các xe khác cùng bến coi như vô dụng.

Giá trị $f(k, i, i + n - 1)$ chính là chi phí ít nhất để đón tất cả các học sinh bởi k xe trong đó xe cuối cùng đón ở bến $i + n - 1$. Đáp số là:

$$\min_{\forall i: 1 \le i \le n} \{f(k, i, i + n - 1\}$$

Thuật toán $O(n^3.k)$ không có gì đặc biệt: Cách tính $f(p, i, j)$ như sau:

- Nếu $p = 1$, $f(1, i, j) = \text{Cost}(i, j)$ theo định nghĩa.
- Nếu $p > 1$ thì xe thứ $p - 1$ sẽ phải đón ở bến t nào đó $\in [i \dots j]$, tất cả các học sinh từ bến $t + 1$ tới j sẽ phải di chuyển tới bến j để lên xe thứ p, vậy:

  $$f(p, i, j) = \min_{\forall t: i \le t \le j} \{f(p - 1, i, t) + \text{Cost}(t + 1, j)\}$$

  Chú ý rằng khi $t = j$, ta coi như xe thứ p không đón học sinh nào cả vì nó ở cùng bến với xe thứ $p - 1$ ($\text{Cost}(t + 1, j) = 0$)

Một phép giải công thức truy hồi trực tiếp tính $f(p, i, j)$ với độ phức tạp $O(j - i)$ là đủ cho subtask 2.

Subtask 3:

Với p và q là hai số nguyên dương, xét hai bến i, j trong đó $1 \le i \le j \le 2n$ và $j \le i + n - 1$. Ta có:

$$f(p + q, i, j) = \min_{\forall t: i \le t \le j} \{f(p, i, t) + f(q, t + 1, j)\}$$

Giải thích: Trong $p + q$ xe thì xe thứ p sẽ phải đón tại bến t nào đó $\in [i \dots j]$. Vậy thì chi phí nhỏ nhất đón đoạn $i \dots j$ bởi $p + q$ xe bằng: (1)

- Chi phí nhỏ nhất đón đoạn $i \dots t$ bởi p xe: $f(p, i, t)$
- Cộng chi phí nhỏ nhất đón đoạn $t + 1 \dots j$ bởi q xe: $f(q, t + 1, j)$

Công thức trên xây dựng nên toán tử $\otimes$ trên các ma trận, xét ba ma trận X, Y và Z trong đó:

$$X[i][j] = f(p, i, j)$$
$$Y[i][j] = f(q, i, j)$$
$$Z[i][j] = f(p + q, i, j)$$

thì công thức (1) cho ta cách tính $Z = Y \otimes Z$, với:

$$Z[i][j] = \min_{\forall t: i \le t \le j} \{X[i][t] + Y[t + 1][j]\}$$

Đến đây có thể hình dung ra thuật toán: Khởi tạo ma trận M trong đó $M[i][j] = f(1, i, j) = \text{Cost}(i, j)$.

Sau đó tính ma trận:

$$\mathbb{M} = M^k = \underbrace{M \otimes M \otimes \ldots \otimes M}_{k \text{ ma trận } M}$$

Phần tử $\mathbb{M}[i][j]$ chính là $f(k, i, j)$.

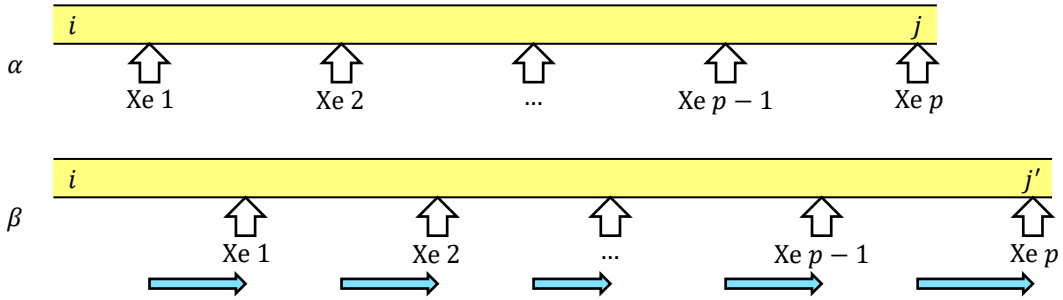
Vì phép tính $\otimes$ trên một dãy các ma trận M có tính kết hợp (suy ra từ định nghĩa của hàm f), ta có thể dùng phương pháp chia để trị giảm độ phức tạp tính toán. Để tránh phiền toái với giới hạn bộ nhớ stack và hàm đệ quy, nên thực hiện bằng thuật toán lặp*.

Phép toán $\otimes$ có thể cài đặt với độ phức tạp $O(n^3)$, ta có thuật toán $O(n^3 \log k)$.

Subtask 4:

Bổ đề 1: Với ba bến bất kỳ $i \le j < j'$.

Xét một phương án tối ưu α đón đoạn $i \ldots j$ bởi p xe. Khi đó tồn tại phương án tối ưu β đón đoạn $i \ldots j'$ bởi p xe mà vị trí của từng xe trong phương án β không nhỏ hơn vị trí của xe đó trong phương án α.

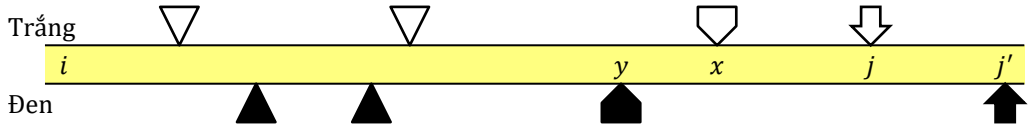


Chứng minh:

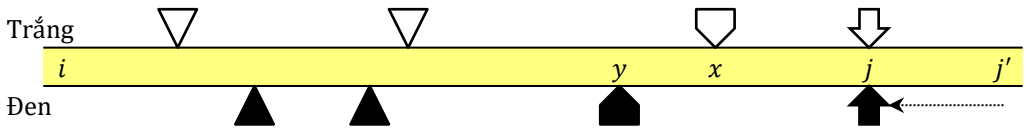
Nhận xét hiển nhiên đúng với xe p: Trong phương án α, xe p phải ở vị trí j còn trong phương án β, xe p phải ở vị trí $j' > j$.

Ta chỉ cần chứng minh bổ đề đúng với xe $p - 1$, tự bổ đề sẽ cho chứng minh quy nạp với các xe khác.

Gọi các xe trong phương án α là xe "trắng". Xét một phương án tối ưu β đón đoạn $i \ldots j'$ bởi p xe màu "đen". Giả sử xe trắng thứ $p - 1$ ở vị trí x còn xe đen thứ $p - 1$ ở vị trí y. Giả sử $y < x$, ta sẽ chỉ ra rằng việc thay toàn bộ các xe đen từ 1 tới $p - 1$ bởi xe trắng sẽ thu được phương án mới ít ra là không tệ hơn so với β.



Đầu tiên ta đặt xe đen cuối cùng lên bến j thay vì j' (xem hình dưới)



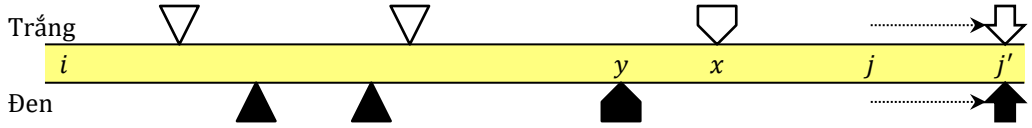
Vì phương án xe trắng là tối ưu để đón đoạn $i \ldots j$ bởi p xe, nên ở tình trạng hiện tại:

---

* Đặt $\mathbb{M} = M = M^1$. Xét lần lượt các chữ số nhị phân của k từ trái qua phải, bỏ qua chữ số đầu tiên (chữ số 1 ở hàng cao nhất). Tại mỗi lượt lặp, đầu tiên "bình phương" ma trận $\mathbb{M}$: $\mathbb{M} = \mathbb{M} \otimes \mathbb{M}$, sau đó nếu chữ số đang xét là 1 thì "nhân" $\mathbb{M}$ với M nữa: $\mathbb{M} = \mathbb{M} \otimes M$

$$\text{Chi phí(Xe trắng)} \le \text{Chi phí(Xe đen)}$$

Bây giờ nếu kéo xe đen và xe trắng cuối cùng không đón ở j nữa mà đón ở j' ($j' > j$), chi phí trên hai phương án sẽ thay đổi như thế nào?



Các học sinh được xếp vào các xe (trắng hoặc đen) từ 1 tới $p-1$ chi phí không thay đổi.

Các học sinh ở bến $x+1 \dots j'$ có quãng đường di chuyển như nhau trên cả hai phương án trắng/đen

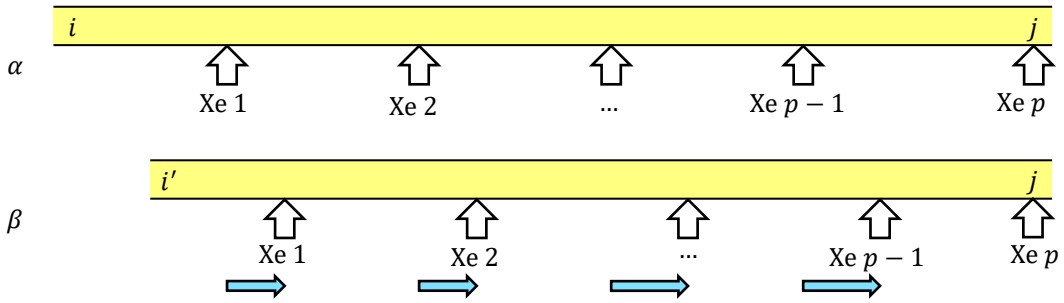
Riêng phương án dùng xe đen, mỗi học sinh ở bến $y+1 \dots x$ bị tăng thêm một đoạn đường di chuyển bằng $j'-j$ (thay vì lên xe ở j thì phải lên xe ở j'). Vậy ta vẫn có:

$$\text{Chi phí(Xe trắng)} \le \text{Chi phí(Xe đen)} = \text{Chi phí}(\beta)$$

Vì β tối ưu, phương án dùng các xe trắng trong trường hợp này cũng phải tối ưu, hay nói cách khác, tồn tại phương án tối ưu đón đoạn $i \dots j$ bởi p xe mà xe thứ $p-1$ ở bến $y \ge x$

Bổ đề 2: Với ba bến bất kỳ $i < i' \le j$.

Xét một phương án tối ưu α đón đoạn $i \dots j$ bởi p xe. Khi đó tồn tại phương án tối ưu β đón đoạn $i' \dots j$ bởi p xe mà vị trí của từng xe trong phương án β không nhỏ hơn vị trí của xe đó trong phương án α.



Phép chứng minh tương tự như bổ đề 1.

Quay lại thuật toán quy hoạch động, xét ba ma trận X, Y và Z trong đó:

$$X[i][j] = f(p,i,j)$$
$$Y[i][j] = f(q,i,j)$$
$$Z[i][j] = f(p+q,i,j)$$

Ta tìm cách tính $Z = Y \otimes Z$ trong thời gian $O(n^2)$, với:

$$Z[i][j] = \min_{\forall t: i \le t \le j} \{X[i][t] + Y[t+1][j]\}$$

Ở đây các ma trận có kích thước $2n \times 2n$, công thức tính ở trên là cho các phần tử $Z[i][j]$ mà $1 \le i \le j < 2n$ và $j < i+n$, nhưng phần tử khác trong ma trận Z coi như bằng 0.

Chiến lược là ta sẽ tính các $Z[i][i]$ trước, sau đó tính các $Z[i][i+1]$, rồi tính các $Z[i][i+2]$...:

```
1 | for (int step = 0; step < n; ++step)
2 |     for (int i = 1; i + step < 2 * n; ++i)
3 |     {
4 |             int j = i + step;
5 |             //Tính Z[i][j];
6 |     }
```

Để tính $Z[i][j]$ xét công thức:

$$Z[i][j] = \min_{\forall t: i \le t \le j} \{X[i][t] + Y[t+1][j]\}$$

Ta lưu lại vết: $trace[i][j]$ bằng giá trị t cho biểu thức vế phải nhỏ nhất, tức là:

$$trace[i][j] = \arg\min_{\forall t: i \le t \le j}\{X[i][t] + Y[t+1][j]\}$$

Bổ đề 1 và 2 cho ta hệ thức:

$$trace[i][j-1] \le trace[i][j] \le trace[i+1][j]$$

Vậy để tính $Z[i][j]$, ta không cần duyệt t từ i tới j mà chỉ cần duyệt t từ $trace[i][j-1]$ tới $trace[i+1][j]$ mà thôi:

```
for (int i = 1; i < 2 * n; ++i) //Tính các Z[i][i] trước
{
    Z[i][i] = 0;
    trace[i][i] = i;
}
for (int step = 1; step < n; ++step)
    for (int i = 1; i + step < 2 * n; ++i)
    {
        int j = i + step; //Tính res[i][j];
        res[i][j] = +∞;
        for (int t = trace[i][j - 1]; t <= trace[i + 1][j]; ++t)
            if (Minimize(Z[i][j], X[i][t] + Y[t + 1][j])) //Cực tiểu hóa được Z[i][j] tại giá trị t này
                trace[i][j] = t; //Lưu vết lại
    }
```

Mẹo giải này mới trông có vẻ như chỉ cải thiện tốc độ chương trình, nhưng thực ra nó làm giảm hẳn độ phức tạp tính toán lý thuyết xuống còn $O(n^2)$. Ta sẽ chỉ ra rằng độ phức tạp của 2 vòng lặp for bên trong (for i và for t) chỉ là $O(n)$

Khi $i = 1$, t chạy từ $trace[1][step]$ tới $trace[2][step+1]$

Khi $i = 2$, t chạy từ $trace[2][step+1]$ tới $trace[3][step+2]$

Khi $i = 3$, t chạy từ $trace[3][step+2]$ tới $trace[4][step+3]$

…

Vì tính tăng dần của dãy: $trace[1][step] \le trace[2][step+1] \le trace[3][step+2] \le \cdots$.

Nên xét tổng thể 2 vòng lặp for i và for t, thì biến t chỉ chạy tăng trong phạm vi từ 1 tới $2n$. ĐPT $O(n)$

Kỹ thuật này trong tối ưu hóa gọi là Knuth Optimization, thuật toán nguyên thủy ban đầu dùng để xây dựng cây nhị phân tìm kiếm tối ưu nhưng sau này được dùng trong nhiều kỹ thuật tối ưu hóa khác nữa. Mô hình ứng dụng của nó là để tính $Z[i][j]$ theo $X[i][t]$ và $Y[t][j]$ xét trên mọi t: $i \le t \le j$ và giá trị t tốt nhất dùng để tính $Z[i][j]$ được lưu vào $trace[i][j]$. Tiêu chuẩn để sử dụng Knuth Optimization là:

$$trace[i][j-1] \le trace[i][j] \le trace[i+1][j]$$

Khi đó ta tính $Z[i][j]$ theo thứ tự: $Z[i][j]$ với $j - i$ nhỏ hơn được tính trước và khi tính $Z[i][j]$, chỉ cần for t chạy từ $trace[i][j-1]$ tới $trace[i+1][j]$. Độ phức tạp để tính tất cả các $Z[i][j]$ là $O(n^2)$ với n là kích thước ma trận.

(*) Tính $Cost(i, j)$ là chi phí dồn tất cả các học sinh ở các bến $\in i \ldots j$ về bến j trong thời gian $O(1)$:

Chi phí để đưa các học sinh ở bến t về bến j ($t \le j$) bằng:

$$a_t \times (j - t)$$

Xét các bến từ i đến j ($i \le j$), chi phí để dồn các học sinh trong phạm vi đó về bến j được tính bằng:

$$\begin{aligned}\text{Cost}(i,j) &= \sum_{t=i}^{j}\left(a_t \times (j-t)\right) \\ &= j \times \left(\sum_{t=i}^{j} a_t\right) - \sum_{t=i}^{j}(a_t \times t)\end{aligned} \tag{2}$$

Dùng hai mảng cộng dồn: $b[0 \dots 2n]$ và $c[0 \dots 2n]$ với ý nghĩa:

$$b_t = a_1 + a_2 + \cdots + a_t = \begin{cases}0, \text{nếu } t = 0 \\ b_{t-1} + a_t, \text{nếu } t > 0\end{cases}$$

$$c_t = a_1 \times 1 + a_2 \times 2 + \cdots + a_t \times t = \begin{cases}0, \text{nếu } t = 0 \\ c_{t-1} + a_t \times t, \text{nếu } t > 0\end{cases}$$

Khi đó công thức (2) có thể tính trong thời gian $O(1)$:

$$\text{Cost}(i,j) = j \times \left(b_j - b_{i-1}\right) + \left(c_j - c_{i-1}\right) \tag{3}$$

Trong trường hợp $i > j$, coi như $\text{Cost}(i,j) = 0$ vì số học sinh trong phạm vi $i \dots j$ bằng 0.