การแข่งขันคอมพิวเตอร์โอลิมปิกระดับชาติ ครั้งที่ 21

ณ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ข้อสอบข้อที่ 2 จากทั้งหมด 3 ข้อ

วันพฤหัสบดีที่ 15 พฤษภาคม 2568 เวลา 08.30 – 13.30 น.

ร้านปลอดภาษี (Duty Free)

ร้านค้าปลอดภาษี (Duty Free) คือร้านขายของในสนามบินที่ขายสินค้าในราคาถูกกว่าปกติ เพราะไม่ต้องเสียภาษีนำเข้าและภาษีมูลค่าเพิ่ม โดยเปิดให้เฉพาะผู้โดยสารขาออกเข้าไปเลือกซื้อสินค้าแบรนด์เนม ของฝาก หรือขนมก่อนขึ้นเครื่อง

ในเขตร้านปลอดภาษีของสนามบินสุวรรณภูมิ มีร้านกระเป๋าแบรนด์เนมชื่อดัง ซึ่งใช้ระบบจองล่วงหน้าแบบวันต่อวัน ลูกค้า N คนสามารถจองกระเป๋าไว้ล่วงหน้าได้<u>คนละหนึ่งใบ</u> แต่ละใบ<u>ไม่ซ้ำแบบ</u>กัน ร้านจึงสามารถเตรียมกระเป๋าที่จองไว้ได้ครบอย่างแน่นอน และมีสินค้าพอดีกับจำนวนลูกค้าทุกคน

ร้านกระเป๋าร้านนี้จะจัดวางกระเป๋าที่สั่งไว้ N ใบบนชั้นวางสินค้า เรียง<u>ตามยาวจากซ้ายไปขวา</u> โดยมีหมายเลข 1 ถึง N กำกับสินค้าแต่ละชิ้นเอาไว้ ดังภาพที่ 1 โดยเริ่มต้นพนักงานจะจัดวางกระเป๋าเรียงแบบใดก็ได้ จากนั้นทางร้านจะปล่อยให้ลูกค้าเข้ามาเลือกซื้อกระเป๋าได้ทีละคนตามลำดับการจองจนครบทุกคน ลูกค้าจะซื้อเฉพาะกระเป๋าที่ตนเองจองไว้เท่านั้น

ภาพที่ 1 การจัดเรียงสินค้าของร้านค้าปลอดภาษีร้านนี้

อย่างไรก็ตาม ด้วยความเร่งรีบเพื่อไม่ให้พลาดเที่ยวบิน ลูกค้าแต่ละคนจะมีระดับความเร่งรีบไม่เท่ากัน กำหนดค่า $\text{max\_allowed\_positions}[i-1]$ แทนจำนวนชั้นวางกระเป๋าที่ลูกค้าคนที่ i สามารถเดินไปถึงได้โดยที่จะไม่ตกเครื่องบิน นั่นคือลูกค้าคนที่ i จะไปหาสินค้าที่ได้จองไว้ ที่บริเวณชั้นวางแรกจนถึงเพียงชั้นวางที่ $\text{max\_allowed\_positions}[i-1]$ เท่านั้น หากพบกระเป๋าที่จองไว้ ลูกค้าจะซื้อทันทีโดยไม่ขยับกระเป๋าใบอื่น ๆ หรือรีบเดินออกจากร้านไปหากไม่พบกระเป๋าที่จองไว้

ก่อนลูกค้าคนถัดไปจะเข้าร้าน พนักงานสามารถนำกระเป๋าที่เหลือทั้งหมดมาจัดวางใหม่บนชั้นวางแต่ละหมายเลขได้อย่างอิสระ (จัดวางกระเป๋าใหม่อย่างไรก็ได้ ภายใต้เงื่อนไขกระเป๋า 1 ใบต่อ 1 ชั้นวาง) การจัดวางใหม่นี้นับเป็นการจัดวางใหม่ 1 รอบ

งานของคุณ (Your Task)

เขียนฟังก์ชันที่มีประสิทธิภาพเพื่อหาว่าพนักงานต้องจัดกระเป๋าบนชั้นวางใหม่อย่างน้อยที่สุดกี่รอบ ที่จะทำให้ลูกค้าทุกคนสามารถซื้อกระเป๋าใบโปรดของตนเองได้ครบถ้วน

ข้อนี้มีรูปแบบการเขียนโปรแกรมที่แตกต่างจากรูปแบบปรกติ ซึ่งจะมีใช้ในการแข่งขันจริง โปรดดูส่วนท้ายของโจทย์ข้อนี้เกี่ยวกับวิธีการใช้งาน

รายละเอียดของโปรแกรม

ผู้เข้าแข่งขันจะต้องเขียนฟังก์ชันต่อไปนี้

```
int minimum_bag_rearrangement_time (std::vector<int> max_allowed_positions)
```

โดยที่พารามิเตอร์ต่าง ๆ คือ

- max_allowed_positions เป็นอาเรย์ของตำแหน่งของกระเป๋าที่มากที่สุดที่คนที่ 1, 2, 3, ..., N ตามลำดับ จะยังซื้อกระเป๋าอยู่
- ฟังก์ชันจะต้องคืนค่าเป็นจำนวนเต็ม เพื่อแสดงจำนวนรอบของการจัดเรียงกระเป๋าที่น้อยที่สุด เพื่อให้ผู้โดยสารทั้ง N คนสามารถซื้อกระเป๋าใบโปรดของตนเองได้ครบทุกคนภายใต้เงื่อนไขข้างต้น

ขอบเขตข้อมูล

- $1 <= N <= 2{,}000{,}000$
- $1 <= \text{max\_allowed\_positions}[i-1] <= N$

ตัวอย่าง

ลองพิจารณาการเรียกใช้งานต่อไปนี้

กรณีตัวอย่างที่ 1

max_allowed_positions: [1, 2, 3, 4, 5]

return: 0

ผลลัพธ์จะได้เป็น 0 เนื่องจากสามารถจัดเรียงกระเป๋าเริ่มต้นตามลำดับดังภาพที่ 2

หมายเลขชั้นวาง	#1	#2	#3	#4	#5
กระเป๋าของลูกค้าคนที่: / ลูกค้าคนที่	1	2	3	4	5
1 (max_allowed_positions[0] = 1)	*				
2 (max_allowed_positions[1] = 2)		*			
3 (max_allowed_positions[2] = 3)			*		
4 (max_allowed_positions[3] = 4)				*	
5 (max_allowed_positions[4] = 5)					*

ภาพที่ 2 ตัวอย่างแสดงการเดินชมสินค้าของกรณีตัวอย่างที่ 1 ที่มีข้อจำกัดการเดินชมเท่ากับ [1, 2, 3, 4, 5]
*เครื่องหมาย * แสดงตำแหน่งของกระเป๋าของลูกค้าคนนั้น ๆ บนชั้นวาง พื้นหลังสีเขียวแสดงสินค้าที่ถูกเยี่ยมชมได้*
แสดงให้เห็นว่าลูกค้าทุกคนสามารถซื้อสินค้าที่ตนเองต้องการได้ทั้งหมดโดยไม่ต้องมีการจัดเรียงใหม่

กรณีตัวอย่างที่ 2

max_allowed_positions: [1, 3, 3, 2, 3]

return: 1

ผลลัพธ์จะได้เป็น 1 เนื่องจากไม่มีวิธีจัดเรียงเริ่มต้นแบบไหนเลยที่ทำให้ลูกค้าทุกคนซื้อได้ครบโดยไม่จัดเรียงใหม่ และสามารถพิสูจน์ได้ว่าต้องจัดเรียงน้อยที่สุด 1 รอบ ดังนี้

- จัดเรียงเริ่มต้นตามลำดับ ดังภาพที่ 3

หมายเลขชั้นวาง	#1	#2	#3	#4	#5
กระเป๋าของลูกค้าคนที่: / ลูกค้าคนที่	1	2	4	5	3
1 (max_allowed_positions[0] = 1)	*				
2 (max_allowed_positions[1] = 3)		*			
3 (max_allowed_positions[2] = 3)					*เดินไม่ถึง
4 (max_allowed_positions[3] = 2)			*เดินไม่ถึง		
5 (max_allowed_positions[4] = 3)				*เดินไม่ถึง	

ภาพที่ 3 ตัวอย่างแสดงการเดินชมสินค้าของกรณีตัวอย่างที่ 2 ที่มีข้อจำกัดการเดินชมเท่ากับ [1, 3, 3, 2, 3]
*เครื่องหมาย * แสดงตำแหน่งของกระเป๋าของลูกค้าคนนั้น ๆ บนชั้นวาง พื้นหลังสีเขียวแสดงสินค้าที่ถูกเยี่ยมชมได้*
แสดงให้เห็นว่าลูกค้าทุกคนสามารถซื้อสินค้าที่ตนเองต้องการไม่ครบถ้วน จึงต้องมีการจัดเรียงใหม่

- เมื่อลูกค้าคนที่ 2 ออกจากร้าน ทำการจัดวางใหม่ 1 รอบ เพื่อให้ลูกค้าคนที่ 3 สามารถเลือกซื้อกระเป๋าที่อยู่ในชั้นวางหมายเลข 2 ได้ ลูกค้าคนที่ 4 สามารถเลือกซื้อกระเป๋าที่อยู่ในชั้นวางหมายเลข 1 ได้ และลูกค้าคนที่ 5 สามารถเลือกซื้อกระเป๋าที่อยู่ในชั้นวางหมายเลข 3 ได้ ดังภาพที่ 4

หมายเลขชั้นวาง	#1	#2	#3	#4	#5
กระเป๋าของลูกค้าคนที่: / ลูกค้าคนที่	4	3	5	- ขายแล้ว	- ขายแล้ว
1 (ซื้อไปแล้ว)					
2 (ซื้อไปแล้ว)					
3 (max_allowed_positions[2] = 3)		*			
4 (max_allowed_positions[3] = 2)	*				
5 (max_allowed_positions[4] = 3)			*		

ภาพที่ 4 ตัวอย่างแสดงการเดินชมสินค้าตัวอย่างที่ 2 หลังจากลูกค้าคนที่ 2 ได้ซื้อสินค้า จึงทำการจัดวางตำแหน่งกระเป๋าใหม่ 1 รอบ
เพื่อให้ลูกค้าคนที่ 3, 4 และ 5 ได้เห็นสินค้าก่อนจะเดินออกจากร้าน
*เครื่องหมาย * แสดงตำแหน่งของกระเป๋าของลูกค้าคนนั้น ๆ บนชั้นวาง พื้นหลังสีเขียวแสดงสินค้าที่ถูกเยี่ยมชมได้*
แสดงให้เห็นว่าลูกค้าทุกคนสามารถซื้อสินค้าที่ตนเองต้องการได้ทั้งหมดโดยต้องมีการจัดเรียงใหม่เพียง 1 รอบ

ข้อกำหนด

หัวข้อ	เงื่อนไข
ระยะเวลาสูงสุดที่ใช้ในการประมวลผล	1 วินาที
หน่วยความจำสูงสุดที่ใช้ในการประมวลผล	1024 MB
คะแนนสูงสุดของโจทย์	100 คะแนน

ข้อมูลเพิ่มเติมเกี่ยวกับชุดทดสอบ

ข้อมูลแนะนำที่เกี่ยวข้องกับชุดทดสอบ มีดังนี้

กลุ่มชุดทดสอบที่	คะแนนสูงสุดของกลุ่มชุดทดสอบนี้	เงื่อนไข
1	4 %	$N <= 10$
2	4 %	$N <= 10{,}000$ และ $max\_allowed\_positions[i]$ เท่ากันทั้งหมด
3	5 %	$N <= 10{,}000$ และ $max\_allowed\_positions$ ถูกเรียงจากน้อยไปมาก
4	7 %	$N <= 10{,}000$ และ $max\_allowed\_positions$ ถูกเรียงจากมากไปน้อย
5	25 %	$N <= 10{,}000$
6	34 %	$N <= 200{,}000$
7	21 %	ไม่มีเงื่อนไขเพิ่มเติม

คำแนะนำในการเขียนโปรแกรม

สำหรับข้อนี้ วิธีการเขียนโปรแกรมและส่งโปรแกรมเข้าสู่ระบบเกรดเดอร์ (Grader) จะเป็นรูปแบบที่แตกต่างจากรูปแบบทั่วไปของการเขียนโปรแกรม ที่ผู้เข้าแข่งขันจะต้องเขียนโปรแกรมในส่วนรับข้อมูลนำเข้าและส่วนแสดงผล โดยในข้อนี้ ผู้เข้าแข่งขันจะต้องเขียนโปรแกรมเฉพาะส่วนการคำนวณที่จำเป็นลงในฟังก์ชันที่กำหนดให้ โดยข้อมูลนำเข้าต่าง ๆ จะสามารถใช้ได้ผ่านพารามิเตอร์ของฟังก์ชัน และผลการคำนวณจะต้องส่งกลับผ่านฟังก์ชันด้วยเช่นกัน โดยที่ผู้เข้าแข่งขันไม่ต้องเขียนโปรแกรมในส่วนรับข้อมูลนำเข้า และไม่ต้องเขียนส่วนแสดงผล

ผู้เข้าแข่งขันจะได้รับไฟล์ต่าง ๆ 4 ไฟล์ดังนี้ ผู้เข้าแข่งขันจะต้องแก้ไขไฟล์เดียวตามที่โจทย์กำหนด และส่งไฟล์ดังกล่าวเข้าสู่ระบบ

- grader.cpp เป็นไฟล์หลักที่ทำหน้าที่รับข้อมูลนำเข้าและส่งออก
- duty_free.cpp เป็นไฟล์ที่ผู้เข้าแข่งขันจะต้องเขียนโปรแกรมคำนวณต่าง ๆ และเป็นไฟล์ที่ผู้เข้าแข่งขันส่งเข้าสู่ระบบ (ในไฟล์ duty_free.cpp ห้ามมีฟังก์ชัน main() แต่มีฟังก์ชันอื่น ๆ ได้)
- run_cpp.sh เป็นไฟล์ที่ผู้เข้าแข่งขันสามารถเรียกใช้เพื่อทำการรันโปรแกรมทั้งหมดได้
- compile_cpp.sh เป็นไฟล์ที่ผู้เข้าแข่งขันสามารถเรียกใช้เพื่อทำการคอมไพล์โปรแกรมได้

ผู้เข้าแข่งขันจะได้รับไฟล์ .zip ซึ่งสามารถดาวน์โหลดได้จากระบบโดยในไฟล์ดังกล่าวเมื่อขยาย (extract) ออกมาแล้วจะมีไฟล์ทั้ง 4 อยู่ ผู้เข้าแข่งขันควรใช้ text editor หรือ IDE ในการแก้ไขไฟล์ duty_free.cpp และทำการคอมไพล์หรือรันโปรแกรมผ่านการเรียกใช้ฟังก์ชัน compile_cpp.sh หรือ run_cpp.sh

เมื่อผู้เข้าแข่งขันสามารถเรียกไฟล์ run_cpp.sh เพื่อทำการรันโปรแกรม โปรแกรมจะทำงานตามที่ได้ระบุไว้ในไฟล์ grader.cpp ซึ่งหัวข้อ “คำอธิบายไฟล์เกรดเดอร์” อธิบายถึงการทำงานของไฟล์ดังกล่าว