**Power of Jump**

เห็นได้ชัดเลยว่าข้อนี้เป็น dp แต่เราจะมาดูทีละ step ๆ กัน

1. ทำแบบเถรตรง O(n2k)

เราจะมาดูวิธีแบบเถรตรง O(n2k) กันก่อนนะครับ

เราจะนิยาม dp[i][j] ว่า “กระโดดมายังช่องที่ i โดยใช้พลังไปแล้ว j หน่วยได้กี่วิธี”

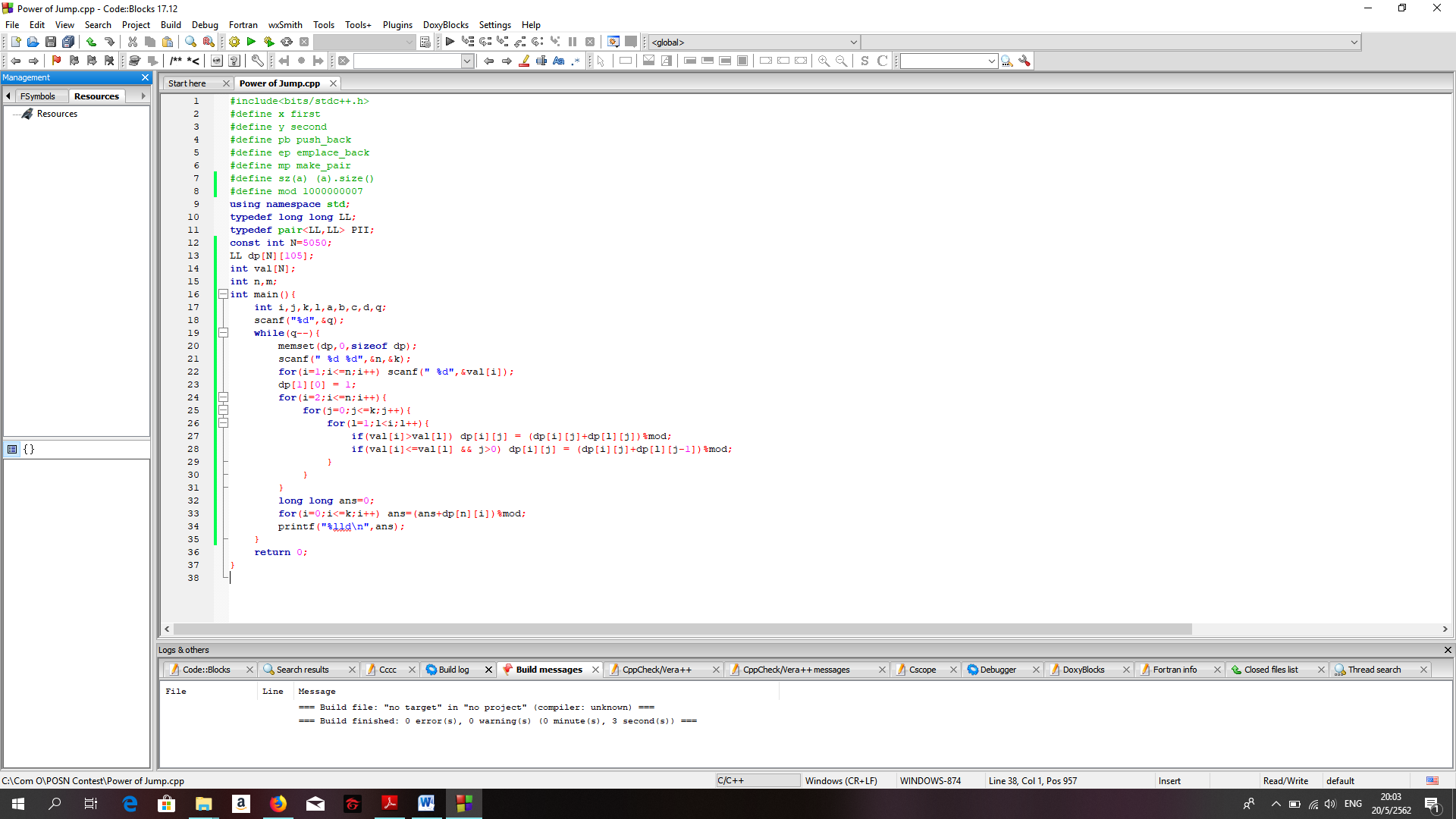
Basecase : dp[1][0] = 1

ทีนี้ เราก็จะมาดูว่า dp[i][j] มาจากอะไรได้บ้างกัน

1. มาได้จากการกระโดดจากช่องก่อนหน้าที่**มีค่าน้อยกว่า** แล้วใช้พลังไปแล้ว j หน่วย หรือก็คือมาจาก dp[l][j] ที่ val[l]<val[i] และ l<i นั่นเอง

2. มาได้จากการกระโดดจากช่องก่อนหน้าโดยใช้พลังจั๊มพ์จากช่องที่**มีค่ามากกว่าเท่ากับ** ที่ใช้พลังไปแล้ว j-1 หน่วย หรือก็คือมาจาก dp[l][j-1] ที่ val[l]>val[i] และ l<i นั่นเอง

โค้ดจะประมาณนี้เลยครับ



แต่เนื่องจากข้อนี้ O(n2k) ช้าเกินไป

เราจะมาดูวิธีการที่คล้ายๆเดิม แต่ทำให้เหลือ O(nklog n) กันครับ

(ใครจะหยุดอ่านตรงนี้ แล้วลองไปคิดต่อก่อนก็ได้นะครับ)

2. วิธี O(nklog n)

สังเกตว่า การที่เราจะกระโดดมาจากช่องก่อนหน้ามาหาเราโดยไม่ใช้พลัง

ต้องเป็นช่องที่มีค่า**น้อยกว่า**เรา

การที่เราจะกระโดดมาจากช่องก่อนหน้ามาหาเราโดยใช้พลัง

ต้องเป็นช่องที่มีค่า**มากกว่าเท่ากับ**เรา

ดังนั้น เราก็ sort ไว้ล่วงหน้าซะเลย!!!!!

เราจะพยายามทำให้เป็นแบบนี้ครับ(นิยามมิกจะเปลี่ยนไปเล็กน้อย)

เวลาเราจะเอาจำนวนวิธีที่กระโดดมาหาเราโดยไม่ใช้พลัง เราก็ดูแค่**ฝั่งซ้าย**ของเรา

เวลาเราจะเอาจำนวนวิธีที่กระโดดมาหาเราโดยใช้พลัง เราก็ดูแค่**ฝั่งขวา**ของเรา

ซึ่งการ sort เพื่อให้เป็นแบบที่เราต้องการจะมีเงื่อนไขประมาณนี้ครับ

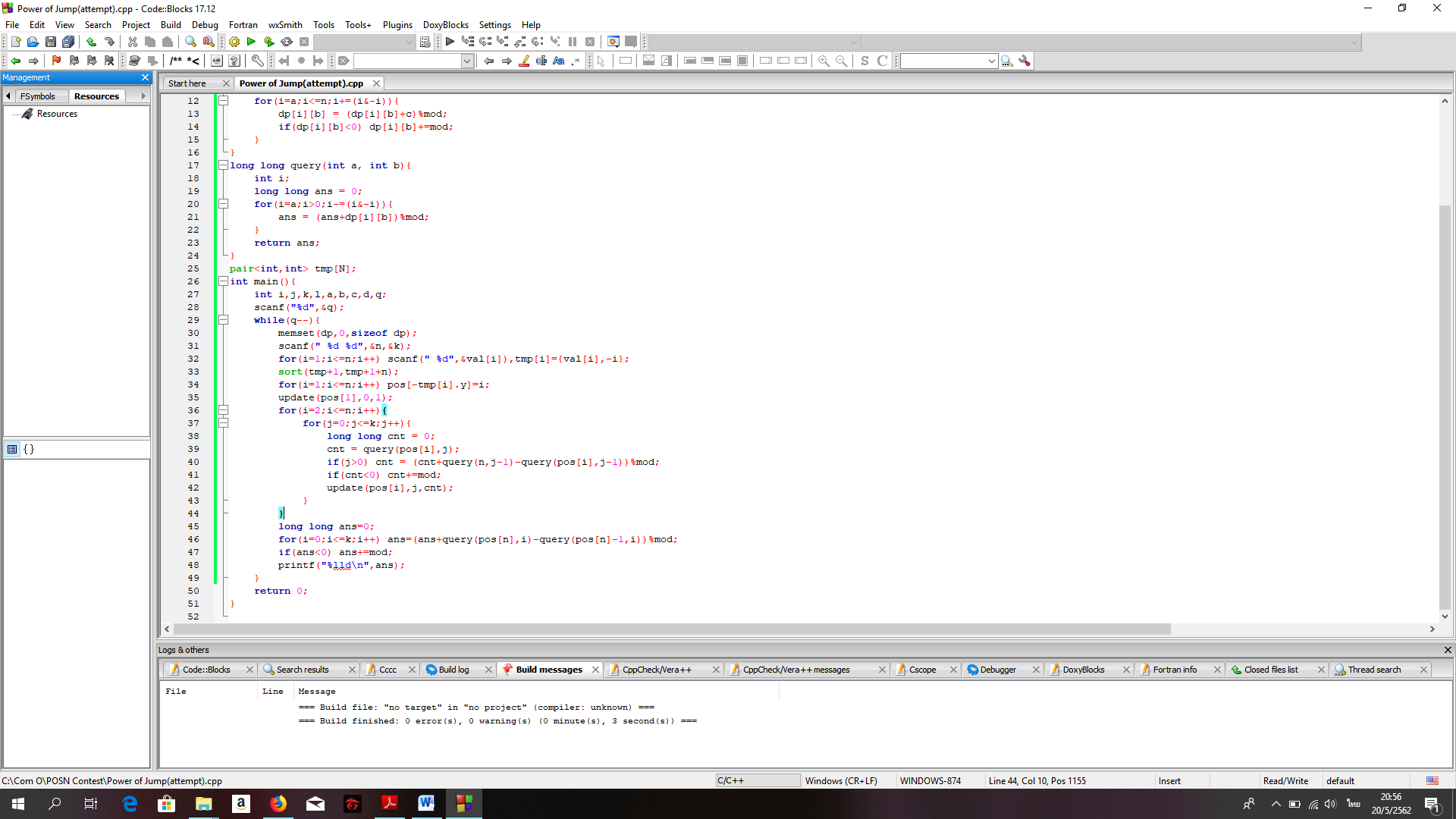
1. ถ้า val[i] != val[j] ให้เอาอันน้อยขึ้นก่อน

2. ถ้า val[i] == val[j] ให้เอาอันที่มี index มากขึ้นก่อน

ซึ่งตรงนี้ เราจะเขียน struct เอง หรือใช้แค่ pair ก็ได้ครับ

หลังจาก sort เสร็จ เราก็ดูว่า index ใหม่ของแต่ละแท่นกระโดดคืออะไร

(ให้ pos[i] คือ index ที่ได้หลังจากการ sort ของแท่นกระโดดที่ i)



นิยามมิกจะเปลี่ยนไปเป็นประมาณนี้

dp[i][j] = จำนวนวิธีในการกระโดดมายัง**แท่นที่ sort แล้วมีลำดับที่ i** แล้วใช้พลังมาทั้งหมด j หน่วย

หรือก็คือ จะพูดอีกนัยหนึ่งว่า dp[ pos[i] ][j] = “กระโดดมายังช่องที่ i โดยใช้พลังไปแล้ว j หน่วยได้กี่วิธี” นั่นแหละ

ทีนี้ เราจะไล่ทำตั้งแต่แท่นที่ 1,2,3,4,…,n ตามลำดับเลยครับ

เรามาดูว่า ตอนเรากำลังทำของ แท่นที่ i

dp[ pos[i] ][j] มาจากอะไรได้บ้าง

1. มาได้จากการกระโดดจากช่องก่อนหน้าที่**มีค่าน้อยกว่า** แล้วใช้พลังไปแล้ว j หน่วย หรือก็คือมาจาก dp[1][j]+dp[2][j]+dp[3][j]+…+dp[pos[i]-1][j] นั่นเอง (ไม่ต้องกลัวว่ามันจะมีช่องที่เป็นของแท่นกระโดดหลังจากเราครับ เพราะ เรายังไม่ได้ update จึงมีค่าเป็น 0)

2. มาได้จากการกระโดดจากช่องก่อนหน้าโดยใช้พลังจั๊มพ์จากช่องที่**มีค่ามากกว่าเท่ากับ** ที่ใช้พลังไปแล้ว j-1 หน่วย หรือก็คือมาจาก dp[val[i]+1][j-1]+dp[val[i]+2][j-1]+dp[val[i]+3][j-1]+…+dp[n][j-1] นั่นเอง(เหมือนเดิม ไม่ต้องกลัวว่ามันจะมีช่องที่เป็นของแท่นกระโดดหลังจากเรา เพราะ เรายังไม่ได้ update จึงมีค่าเป็น 0)

ทีนี้ ถ้าเราไล่วนลูปหา dp[1][j]+dp[2][j]+dp[3][j]+…+dp[pos[i]-1][j] หรือ dp[val[i]+1][j-1]+dp[val[i]+2][j-1]+dp[val[i]+3][j-1]+…+dp[n][j-1]

มันก็จะเสียเวลาเท่ากับวิธีเดิม คือ O(n)

มีวิธีหาค่าทั้งสองอย่างนั้นแบบเร็วๆมั้ยนะ....?

ใช่แล้วครับ Fenwick Tree นั่นเอง!!!!!!!

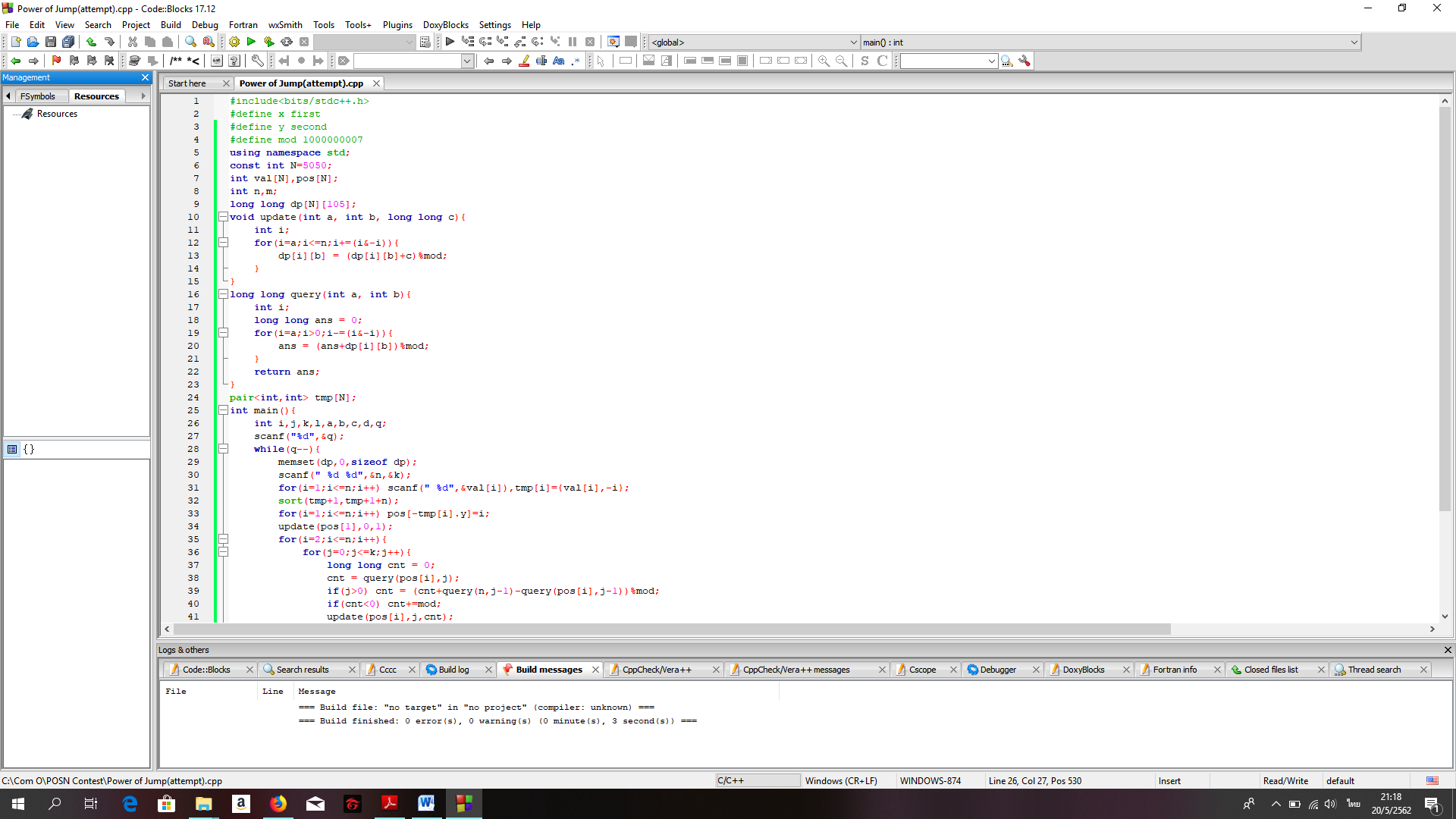
เราก็ทำให้ dp[…][0],dp[…][1],dp[…][2], … , dp[…][k] เป็นต้น fenwick ซะเลย

ที่นี้ เราก็จะหาค่า dp[1][j]+dp[2][j]+dp[3][j]+…+dp[pos[i]-1][j] หรือ dp[val[i]+1][j-1]+dp[val[i]+2][j-1]+dp[val[i]+3][j-1]+…+dp[n][j-1] ภายใน log n แล้ว!!!!!!!!!!!!

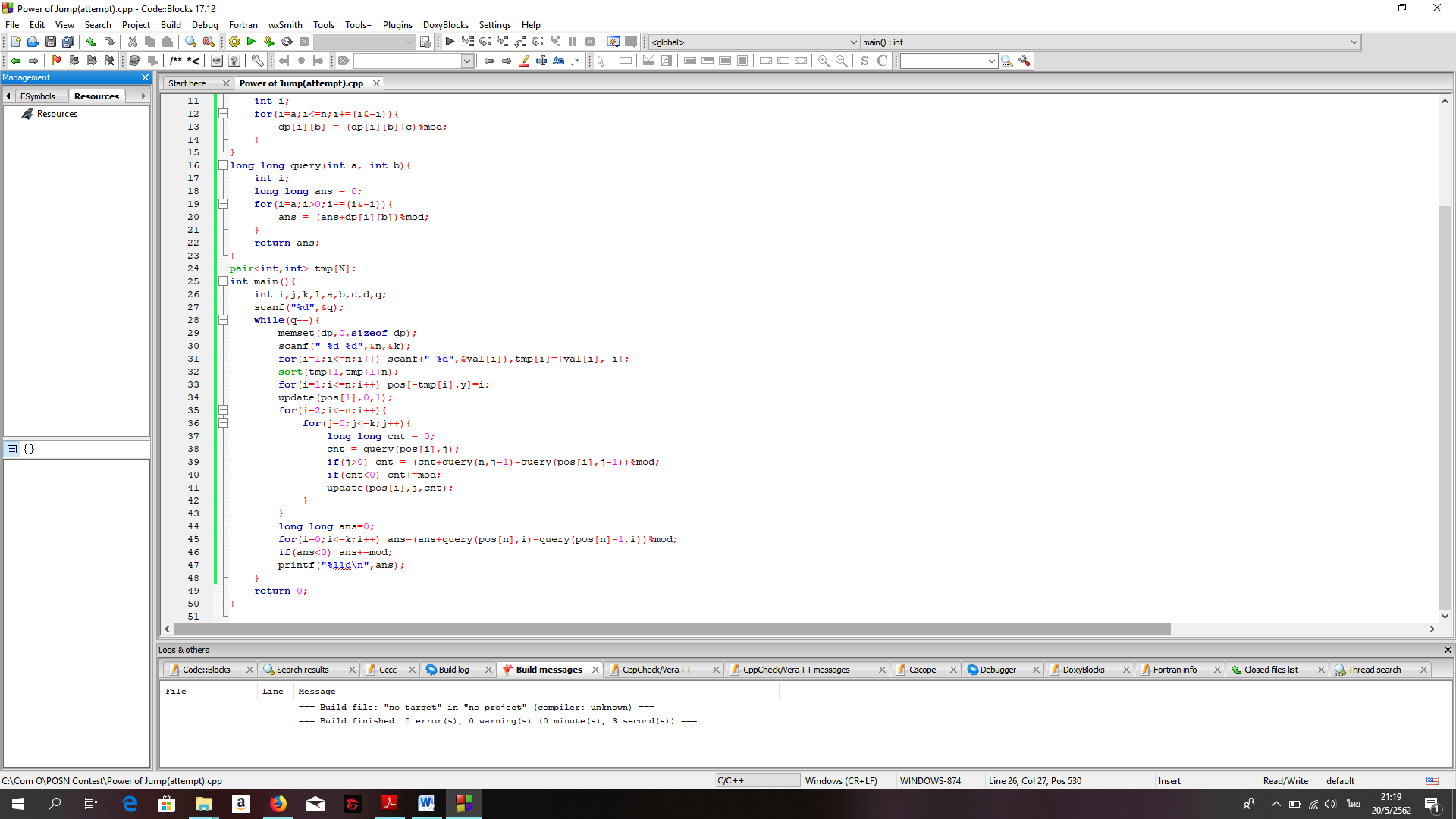
อาจจะเขียนยากซักหน่อย แต่โค้ดของต้น fenwick จะประมาณนี้เลยครับ( mod ให้ดีๆนะครับ )

Update(a,b,c) คือ update dp[a][b] ให้มีค่าเพิ่มขึ้น c

Query(a,b) คือ หาว่า dp[1][b]+dp[2][b]+…+dp[a][b] มีค่าเท่าไร



ส่วนโค้ดของ dp ที่เหลือก็แบบนี้เลยครับ



Happy Coding!!!!