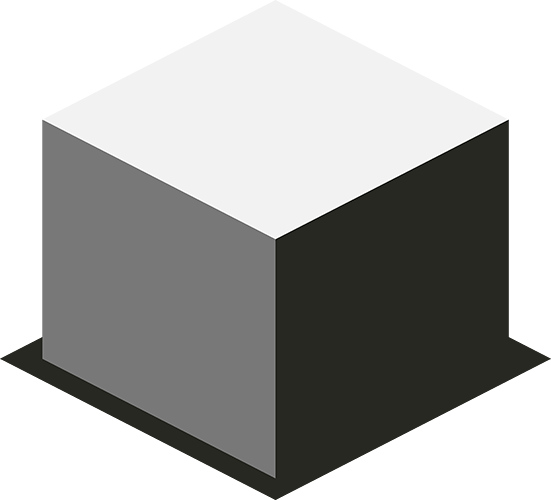


**PreTOI14**

**Editorial**

 Destruction by AquaBlitz11

โจทย์ข้อนี้เป็นโจทย์ Dynamic Programming ที่ค่อนข้างยากข้อหนึ่ง ก่อนอื่น ต้องสังเกตว่าการเลือกต่อยโบราณสถานให้เหลือผลรวมส่วนที่เหลือมากที่สุด สามารถแก้ได้โดยพิจารณาหาวิธีการต่อยโบราณสถานให้ผลรวมส่วนที่ต่อยไปน้อยที่สุด แล้วนำคำตอบที่ได้มาลบออกจากผลรวมทั้งหมด

**1. กรณี** 

สังเกตว่าในกรณีที่  เราต้องหาช่วงที่มีผลรวมน้อยที่สุดเพียงช่วงเดียว ซึ่งคล้ายคลึงกับโจทย์คลาสสิก Maximum Sum Subarray ที่ให้หาผลรวมมากที่สุด หากจำวิธีแก้ปัญหานี้ได้ ก็จะได้คะแนน 15 คะแนนในชุดทดสอบนี้ (หากคิดวิธีกรณีทั่วไปออก ก็ไม่จำเป็นต้องเสียเวลา implement ส่วนนี้)

ปัญหานี้สามารถแก้ได้ใน  โดยใช้วิธี Dynamic Programming

นิยามให้  เท่ากับผลรวมที่น้อยที่สุดที่เป็นไปได้ เมื่อพิจารณา subarray ทุกแบบที่จบที่ตำแหน่ง  จะได้ Recurrence Relation คือ  แต่จะได้คำตอบสุดท้ายของปัญหาเป็น  เพราะเราอนุญาตให้แต่ละช่วงจบที่ตำแหน่งใดก็ได้

สามารถศึกษาเพิ่มเติมได้ที่ <https://www.geeksforgeeks.org/largest-sum-contiguous-subarray/>

**2. กรณี** 

เนื่องจาก  เล็กพอ เราอาจจะใช้วิธี Brute Force ได้ โดยทดลอง Subset ของโบราณสถานที่ถูกต้องทิ้งทั้งหมดที่เป็นไปได้ แล้วลูปตรวจสอบว่าตรงเงื่อนไขที่กำหนดให้หรือไม่ หากตรงเงื่อนไขก็ให้เก็บคำตอบที่ดีที่สุดเอาไว้ วิธีนี้ทำงานใน  หากทำวิธีนี้จะได้ 15 คะแนน

**3. กรณี **

แก้ได้ด้วย Dynamic Programming โดยกำหนดนิยาม state ดังนี้

 เท่ากับ ผลรวมส่วนที่ถูกต่อยที่น้อยที่สุดที่เป็นไปได้ เมื่อต้องต่อย  ครั้ง และพิจารณา array เพียงตำแหน่งที่ 1 ถึง  เท่านั้น สำหรับ Base Case กำหนดให้  สำหรับทุก  เท่ากับ 0 (เนื่องจากว่าต้องต่อยอีก 0 ครั้ง) และกำหนดให้  สำหรับทุก  เท่ากับ  เนื่องจากเราต้องต่อยอีก  ครั้ง แต่ช่วงที่เราเลือกต่อได้เหลือเพียงแค่ 0

เมื่อกำหนดนิยามแล้ว การหาค่าของ  เราจะพิจารณาตำแหน่ง  เป็นหลัก โดยจะเลือกวิธีที่สุดจากวิธีดังต่อไปดังนี้

1) เลือกไม่ต่อยช่วงที่เกี่ยวข้องกับตำแหน่งที่  ดังนั้น ปัญหาก็จะลดเหลือเพียงส่วน 

2) เลือกต่อยหนึ่งช่วง โดยช่วงเริ่มต้นที่ตำแหน่ง  (ทดลองทุกค่า  ที่เป็นไปได้) จบที่ตำแหน่ง  และขนาดช่วงต้องมากกว่าหรือเท่ากับ  โดยเมื่อเลือกต่อยช่วงนั้นไปแล้ว จะทำให้เกิดมูลค่าเสียหายเท่ากับ  และต้องพิจารณาการต่อย  ครั้งที่เหลือใน array  ตัวทางด้านซ้าย รวมแล้วจะเกิดมูลค่าเสียหายทั้งหมด  (ถ้า  ให้ถือว่า )

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |

**เว้น**

**ช่วงที่เลือกต่อย**

 **ตัวที่ยังต้องต่อยเพิ่มอีก**  **ครั้ง**

ดังนั้น สามารถสรุปเป็นสมการได้ว่า



เนื่องจากว่าตาราง  มีขนาด  ในแต่ละช่องเราต้องลูปหาค่า  ที่ดีที่สุดเป็นเวลา  โดยต้องลูปหาผลรวม  ใน  อีกชน ดังนั้น สุดท้ายแล้ว จะได้ Time Complexity เท่ากับ  และ Space Complexity เป็น 

สุดท้าย จะได้ว่ามูลค่าส่วนที่ถูกต่อยไปที่น้อยที่สุดจะเท่ากับ  แต่อย่าลืมว่าเราต้องการมูลค่าส่วนที่เหลืออยู่ ดังนั้นต้องนำผลรวมมูลค่าทั้งหมดมาลบจึงจะได้คำตอบ

**4. กรณี **

ใช้สูตรตาม Subtask 3 แต่ลดเวลาในการหาผลรวมโดยสร้าง Partial Sum Array  ก่อน นิยามให้  เราสามารถสร้าง array  ได้ใน  จากความสัมพันธ์ 

หากต้องการหาผลรวม  สามารถหาได้จาก  ดังนั้น จะได้ส่วนที่สองของสมการเวียนเกิดเป็น  รวมแล้ว Time Complexity ของอัลกอริทึมทั้งหมดจะเป็น 

**5. กรณี **

สังเกตว่า ส่วนที่สองของสมการเวียนเกิดสามารถแยกเป็น  โดยส่วนที่เราต้องการหาค่า  นั้นมีการคำนวณซ้ำซ้อนมากเกินไป เพราะเมื่อขยับ  ไปทางขวาค่านึง ตัวที่ต้องพิจารณาใน  มีเพิ่มมาเพียงตัวเดียวเท่านั้น เราไม่จำเป็นต้องไปลูปเริ่มตรงส่วนก่อนหน้าใหม่ก็ได้

เพื่อความสะดวกในการเขียนสมการเวียนเกิด ขอกำหนดนิยามของตาราง  (Prefix Min.) เป็น 

เช่นเดียวกับการสร้าง Partial Sum Array เราสามารถสร้างตาราง  ได้จากความสัมพันธ์  สุดท้ายแล้ว จากเดิมที่สมการเวียนเกิดเป็น



เราสามารถเขียนใหม่ได้ ดังนี้



สังเกตว่าในแต่ละช่องของตาราง  เราใช้เวลาหาคำตอบเพียง  เท่านั้น ดังนั้น โดยรวมแล้ว อัลกอริทึมทั้งหมดนี้จะใช้เวลาเพียง 

**6. กรณีทั่วไป ()**

ปัญหาที่เกิดขึ้นในครั้งนี้ไม่ใช่เรื่องเวลา แต่เป็นเรื่องความจำแทน เพราะหากเราใช้ตาราง  ขนาด  จะต้องเสียพื้นที่มากถึง  (ใช้ตัวแปรประเภท long long)

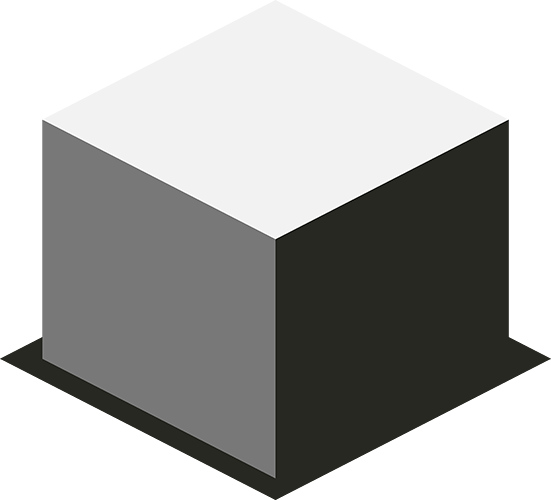
สังเกตว่าสมการเวียนเกิดดังกล่าวอ้างอิงถึงข้อมูลในแถวที่  (แถวก่อนหน้า) และแถวที่  (แถวปัจจุบัน) เท่านั้น ดังนั้น เราเก็บเพียงแค่สองแถวสุดท้ายก็ได้ โดยเก็บเป็นตารางเพียง 2 แถว โดยให้แถวบนแทนแถวที่  และแถวล่างแทนแถวที่ 

วิธีนี้จะลด Space Complexity เหลือ  ซึ่งทำให้ได้คะแนนเต็มในข้อนี้

(Solution Code อยู่ในหน้าถัดไป)

**Solution Code**

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  using ll = long long;  const ll INF = 1e17;  const int N = 100010;  const int M = 100010;  const int K = 1010;  ll A[N], qs[N], dp[2][N], pmin[2][N];  int main()  {  int n, k, m;  scanf("%d%d%d", &n, &k, &m);  pmin[0][0] = INF;  for (int i = 1; i <= n; ++i) {  scanf("%lld", &A[i]);  qs[i] = qs[i-1]+A[i];  pmin[0][i] = min(pmin[0][i-1], -qs[i-1]);  }    for (int i = 1; i <= k; ++i) {  int x = i&1;  dp[x][0] = INF;  pmin[x][0] = INF;  for (int j = 1; j <= n; ++j) {  dp[x][j] = dp[x][j-1];  if (j-m+1 >= 1)  dp[x][j] = min(dp[x][j], pmin[x^1][j-m+1] + qs[j]);  pmin[x][j] = min(pmin[x][j-1], dp[x][max(j-2, 0)] - qs[j-1]);  }  }  printf("%lld\n", qs[n]-dp[k&1][n]);  return 0;  } |

 Copying is not permitted by Plurm2545

ก่อนอื่น สังเกตว่าสิ่งที่โจทย์ข้อนี้ให้มาคือกราฟ โดยมีคนเป็น node และความสัมพันธ์เป็น edge

สิ่งที่ง่ายที่สุดที่เราทำได้คือทดลองทุกค่า  ที่เป็นไปได้ แล้วเช็คว่าทุกคู่ที่พยายามจะลอกกันยังอยู่ใน component เดียวกันหรือไม่ โดยเก็บค่า  ที่มากที่สุดที่ตรงเงื่อนไขไว้ การเช็คมีสามวิธีหลัก ๆ ดังนี้

1. สำหรับแต่ละคู่ เราจะทำการ Depth-first Search เริ่มจากคน ๆ หนึ่ง (พิจารณาเฉพาะ edge ที่ยังไม่ถูกตัด) เพื่อเช็คว่าสามารถไปถึงอีกคนหนึ่งได้หรือไม่ เนื่องจากว่ามีมากสุดถึง  คู่ แต่ละคู่ใช้เวลาในการเช็คมากสุด  ดังนั้น หากรวมกับการทดลองทุกค่า  ที่เป็นไปได้ จะต้องใช้เวลามากถึง  ซึ่งจะได้คะแนนมากสุดเพียง 20 คะแนนเท่านั้น

2. เราจะทำการระบุหมายเลข component ให้กับ node แต่ละ node จนครบก่อน โดยทำการ DFS จากจุดหนึ่งไปถึงทุก ๆ จุด (พิจารณาเฉพาะ edge ที่ยังไม่ถูกตัด) แล้วกำหนดให้ทุกจุดที่ไปถึงได้เป็นหมายเลขเดียวกัน และหากมีจุดไหนที่ยังไปไม่ถึง ก็ให้ไปเริ่มจากจุดพวกนั้นต่อ โดยเปลี่ยนหมายเลข component เรื่อย ๆ เมื่อทำครบแล้วจึงเช็คแต่ละคู่ว่าอยู่ component เดียวกันหรือไม่ โดยรวมแล้วค่า  หนึ่งค่า จะทำให้เกิดการ DFS บน  node  edge เท่านั้น ไม่มีการ DFS ซ้ำ node เดิมเหมือนวิธีแรก ส่วนการเช็คแต่ละคู่ว่าอยู่ component เดียวกันหรือไม่ สามารถเช็คได้ใน  ดังนั้น Time Complexity รวมจึงเป็น  แต่จะได้คะแนนมากสุดเพียง 45 คะแนนเท่านั้น

3. ใช้ data structure ที่ชื่อ Union-find Disjoint Set ในการหาว่าแต่ละคู่อยู่ component เดียวกันหรือไม่ วิธีนี้จะได้ 45 คะแนนเช่นกัน รายละเอียดการใช้ DSU ให้อ่านที่ Subtask 3 ของข้อ Worker (PreTOI14 #2) ของ Editorial นี้ (หน้าที่ 11)

การที่จะได้คะแนนเต็มได้นั้น ต้องสังเกตว่า ความจริงแล้วเราไม่จำเป็นต้องทดลองค่า  ทั้งหมดที่เป็นไปได้ก็ได้ เพราะมีคุณสมบัติอยู่สองข้อ คือ

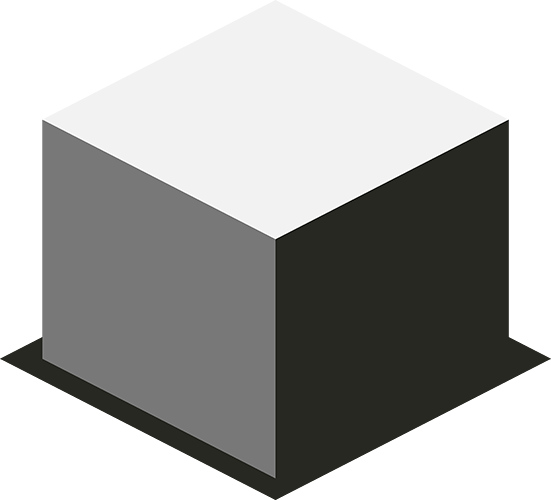
1. ถ้ากำหนดค่า  แล้วทุกคู่ลอกกันไม่ได้ นั่นแปลว่าเราไม่จำเป็นต้องทดลองค่า  ที่น้อยกว่านี้อีกแล้ว (เพราะถ้า  น้อยกว่านี้ ทุกคู่ก็ยังคงลอกกันไม่ได้อยู่ดี และเราพยายามทำให้ค่า  มาก ๆ ด้วย)

2. ถ้ากำหนดค่า  แล้วมีอย่างน้อยหนึ่งคู่ที่ลอกกันได้ นั่นแปลว่าเราต้องใช้ค่า  น้อยกว่านี้ (เพราะเราจำเป็นต้องตัดเพิ่มอีก ถ้าเพิ่มค่า  ไป มีแต่จะทำให้จำนวนคู่ที่ลอกกันได้เพิ่มขึ้น)

ดังนั้น เราสามารถ binary search บนค่า  แล้วจำกัดช่วงตามเงื่อนไข 2 ข้อที่กล่าวมาได้ วิธีนี้จะทำให้ Time Complexity ลดลงเหลือ 

**Solution Code**

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  using pii = pair<int, int>;  const int N = 200010;  const int M = 500010;  int n, m, p, num[N];  bool vis[N];  vector<pii> G[N];  pii P[M];  void dfs(int u, int k, int x) {  vis[u] = true;  num[u] = x;  for (auto v : G[u])  if (v.second < k && !vis[v.first])  dfs(v.first, k, x);  }  bool check(int k) {  fill(vis, vis+n+1, false);  for (int i = 1; i <= n; ++i)  if (!vis[i])  dfs(i, k, i);  for (int i = 0; i < p; ++i)  if (num[P[i].first] == num[P[i].second])  return false;  return true;  }  int main() {  scanf("%d%d%d", &n, &m, &p);  for (int i = 0; i < m; ++i) {  int u, v, w;  scanf("%d%d%d", &u, &v, &w);  G[u].emplace\_back(v, w);  G[v].emplace\_back(u, w);  }  for (int i = 0; i < p; ++i)  scanf("%d%d", &P[i].first, &P[i].second);  int b = 1;  int e = 1e9+1;  while (b < e) {  int mid = (b+e+1)/2;  if (check(mid)) b = mid;  else e = mid-1;  }  if (e == 1e9+1) e = -1;  printf("%d\n", e);  return 0;  } |

 Maximum Submatrix Sum by win11905

สำหรับโจทย์ข้อนี้ จะเน้นสอนเทคนิค Partial Sum Array (หรือบางคนอาจจะเรียกว่า Quick Sum Array) ซึ่งถือได้ว่าเป็นเทคนิคพื้นฐานอย่างหนึ่ง ถึงอย่างไรก็ตาม รู้แต่เทคนิคนี้ก็ไม่พอ เพราะการที่จะได้คะแนนเต็มนั้นต้องใช้การสังเกตลักษณะของเมทริกซ์เพื่อเวลาและความจำที่ต้องใช้ในการประมวลผล

**Subtask 1 (20 คะแนน) -** 

สร้างตารางขึ้นมาใน  แล้วลูปหาสี่เหลี่ยมขนาด  ที่มีผลรวมมากที่สุด โดยการลูปให้ยึดมุมใดมุมหนึ่งเป็นหลักแล้วลูปหาผลรวมของสมาชิกทุกตัวในสี่เหลี่ยม ในกรณีแย่สุดอาจจะมี Time Complexity เป็น  และ Space Complexity เป็น 

**Subtask 2 (30 คะแนน) -** 

สังเกตว่าเราไม่จำเป็นต้องลูปหาผลรวมของสมาชิกทุกตัวก็ได้ แต่ใช้การสร้างเมทริกซ์ Partial Sum  ขนาด  ขึ้นมา โดย  จะเท่ากับผลรวมของสมาชิกในแถวที่ 1 ถึง  คอลัมน์ที่ 1 ถึง  ยกตัวอย่าง  จะมีค่าเท่ากับผลรวมของสมาชิกในช่องที่เน้นสีเหลืองไว้ตามภาพทางด้านซ้าย

การหาค่า  แต่ละช่อง เราไม่จำเป็นต้องลูปหาผลรวมในช่วง แต่เราจะใช้ข้อสังเกตที่ว่า



สังเกตว่าเราใช้  เฉย ๆ ไม่ได้ เพราะจะมีส่วนที่ซ้อนทับกัน (สีเขียวเข้มในภาพทางด้านขวา) จึงต้องลบส่วน  ทิ้ง

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |
|  | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |

เมื่อสร้างเมทริกซ์  ได้แล้ว เราสามารถหาผลรวมของ Submatrix ใด ๆ ได้ใน  โดยผลรวมของสี่เหลี่ยมแถวที่  ถึง  คอลัมน์ที่  ถึง  จะเท่ากับ 

โดยรวมแล้ว วิธีนี้จะทำให้ Time Complexity ลดลงเป็น 

**Subtask 3 (20 คะแนน)** - 

Space Complexity  จะไม่สามารถใช้ได้ เพราะการสร้างตาราง  ขึ้นมาจะต้องใช้เนื้อที่มากถึง  ซึ่งเกิน Memory Limit ที่กำหนดไว้ ดังนั้น หากต้องการใช้วิธีเดิม เราต้องหาผลรวมของ Submatrix ได้โดยไม่ต้องสร้างเมทริกซ์ขึ้นมา

เราจะทดลอง Submatrix Submatrix ขนาด  ทั้งหมดที่เป็นไปได้เหมือนเดิม (ลูปยึดจุดมุมใดมุมหนึ่ง) สมมุติว่า Submatrix ครอบคลุมแถวที่  ถึง  คอลัมน์ที่  ถึง  เราสามารถหาผลรวมได้จาก  เช่น ในภาพด้านล่าง ผลรวมเท่ากับ 

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |

อนึ่ง ในการหาผลรวมของ subarray  เราสามารถทำได้โดยเก็บ Partial Sum คล้าย ๆ วิธีของ Subtask 2 กล่าวคือ  โดยเราสามารถสร้าง array  ได้จากความสัมพันธ์

 ส่วนการหาผลรวมตั้งแต่ตัวที่  ถึง  สามารถหาได้จาก 

(array  ก็เหมือนกัน)

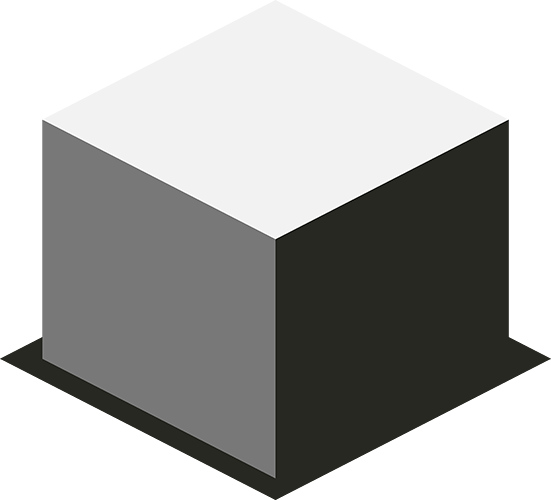
เนื่องจากเราไม่ได้สร้างตารางแล้ว จะทำให้ Space Complexity ลดลงเหลือ  แต่ Time Complexity ยังคงเป็น  อยู่

**Subtask 4 (30 คะแนน) -** 

สังเกตว่า ความจริงแล้วเราไม่จำเป็นต้องทดลองทุกสี่เหลี่ยมที่เป็นไปได้ เพราะผลรวมทั้งหมด ขึ้นอยู่กับผลรวมของ subarray ของ  และ  ที่เลือกมา ซึ่งแยกจากกันโดยสิ้นเชิง ดังนั้น เราเพียงแค่ทดลองหา subarray ขนาด  ของ  ที่ดีที่สุด และหา subarray ขนาด  ของ  ที่ดีที่สุด แล้วนำมารวมกันโดยใช้ความสัมพันธ์  ก็เพียงพอแล้ว วิธีนี้ Time Complexity จะเป็น 

**Solution Code**

|  |
| --- |
| #include <bits/stdc++.h>  #define long long long  using namespace std;  const int N = 1e5+5;  int n, w, h;  long A[N], B[N];  int main() {      scanf("%d %d %d", &n, &w, &h);      for(int i = 1; i <= n; ++i) scanf("%lld", A+i), A[i] += A[i-1];      for(int i = 1; i <= n; ++i) scanf("%lld", B+i), B[i] += B[i-1];      long mx1 = 0, mx2 = 0;      for(int i = w; i <= n; ++i) mx1 = max(mx1, A[i] - A[i-w]);      for(int i = h; i <= n; ++i) mx2 = max(mx2, B[i] - B[i-h]);      printf("%lld\n", mx1 \* h + mx2 \* w);  } |

 Plagiarist by win11905

โจทย์ข้อนี้เป็นหนึ่งในโจทย์คลาสสิกที่ให้ความสำคัญกับ Data Structure เป็นอย่างมาก โดย Data Structure ที่ใช้ในโจทย์ข้อนี้ถือว่าค่อนข้างซับซ้อน แต่สามารถนำไปประยุกต์ใช้ในโจทย์อื่น ๆ ได้จำนวนมาก

**Subtask 1 ถึง 3 (45 คะแนน) -** 

เราสามารถจำลองเหตุการณ์ตามที่โจทย์ระบุได้เลย โดยหากใช้ array ก็จำเป็นจะต้องเลื่อนข้อมูลที่อยู่ทางด้านขวาตัวที่เอาออกเข้ามาเอง หรือหากใช้ std::vector ก็อาจจะใช้ฟังก์ชัน erase ซึ่งเลื่อนตำแหน่งให้โดยอัตโนมัติ ทั้งสองวิธีจะมี Time Complexity เป็น 

**Subtask 4 (25 คะแนน) - **

แทนที่จะจำลองเหตุการณ์โดยการเลื่อนสมาชิกใน array เข้ามา เราจะใช้วิธีเก็บ Boolean Array โดยแต่ละช่องจะระบุว่าข้อมูลช่องนั้นถูกนำออกแล้วหรือยัง เมื่อเราต้องกาหารตัวที่  ที่ยังไม่ถูกนำออก ให้ลูปจากทางซ้ายมาทางขวา นับเฉพาะช่องที่ไม่ได้นำออก จนกว่าจะถึงช่องที่  ถึงอย่างไรก็ตาม ก็ยังช้าเกินอยู่ดี

ให้แบ่ง array ขนาด  ออกเป็น  บล็อก แต่ละบล็อกจะเก็บข้อมูลเพื่อนับว่าในบล็อกนั้นมีข้อมูลกี่ตัวที่ยังไม่ถูกนำออก เมื่อเราต้องการหาตัวที่  ให้เราลูปพิจารณาทีละบล็อกไปก่อน จนกว่าจะเจอบล็อกที่มีข้อมูลตัวที่  อยู่ จึงลูปตามสมาชิกทีละตัวในบล็อกนั้นเพื่อหาตัวที่ 

ยกตัวอย่าง หากมีข้อมูล  ตัว แล้วเราแบ่งออกเป็น  บล็อก บล็อกละ 4 ตัว โดยข้อมูลตัวที่ 3, 6, 10, 11 ถูกนำออกไปแล้ว เราจะเก็บข้อมูลในแต่ละบล็อกดังนี้

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| บล็อกที่ 1: เหลือ 3 ตัว | | | | บล็อกที่ 2: เหลือ 3 ตัว | | | | บล็อกที่ 3: เหลือ 2 ตัว | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

หากต้องการหาตัวที่  ลำดับการทำงานจะเป็นดังนี้

* บล็อกที่ 1 มีเพียง 3 ตัวที่เหลืออยู่ ซึ่งน้อยไป จึงพิจารณาบล็อกถัดไป
* บล็อกที่ 2 มีเพิ่มมาอีก 3 ตัว รวมเป็น 6 ตัว ดังนั้นตัวที่ 5 ต้องอยู่ในบล็อกนี้
* ก่อนหน้าบล็อกนี้มีอยู่แล้ว 3 ตัว ตำแหน่งที่ 5 จึงเป็นตัวที่ 4
* ตำแหน่งที่ 6 ไม่พิจารณาเพราะถูกนำออกแล้ว
* ตำแหน่งที่ 7 เป็นตัวที่ 5 สิ้นสุดการทำงาน

ในกรณีแย่สุด ข้อมูลที่เราต้องการอาจจะอยู่ในบล็อกสุดท้าย รวมแล้วต้องพิจารณาบล็อกทั้งหมด บล็อก และภายในบล็อกสุดท้ายต้องพิจารณาข้อมูลอีก  ตัว ดังนั้น เราควรกำหนดจำนวนบล็อก  เป็น  (โดยประมาณ) เพราะจะทำให้มีจำนวนบล็อกและจำนวนข้อมูลโดด ๆ ที่ต้องพิจารณาเท่ากัน วิธีนี้จะทำให้จำนวนครั้งที่ต้องพิจารณาน้อยสุดเท่าที่เป็นไปได้ โดยพิจารณาเพียง  รอบ

เนื่องจากโจทย์กำหนดให้ดำเนินการทั้งหมด  ครั้ง ดังนั้น Time Complexity จะเป็น  เทคนิคการแบ่ง array เป็น  บล็อกดังที่กล่าวมา เรียกว่า Square Root Decomposition

สามารถศึกษาข้อมูลเพิ่มเติมได้ที่ <https://www.geeksforgeeks.org/sqrt-square-root-decomposition-technique-set-1-introduction/>

**Subtask 5 (30 คะแนน) -** 

1. Order Statistics Tree

ใช้ Binary Search Tree ในการเก็บตำแหน่งข้อมูลที่ยังไม่ถูกนำออก และในแต่ละ node ของ BST จะต้องเก็บจำนวน node ทั้งหมดใน Subtree นั้นด้วย ซึ่งจะต้องอัพเดทเมื่อมีการลบสมาชิกออกจาก BST

หากต้องการหาตัวที่  ให้เริ่มท่องต้นไม้จาก Root ของ BST ก่อน หากจำนวน node ใน subtree ด้านซ้ายมี  node พอดี นั่นแปลว่า node ที่เราพิจารณาอยู่เป็น node ที่  จึงตอบ node ปัจจุบัน

หากจำนวน node ใน subtree ด้านซ้ายมี  node ขึ้นไป นั่นแปลว่า node ที่  อยู่ทางด้านซ้าย จึงต้องท่องต้นไม้ต่อไปทางด้านซ้าย แต่หากจำนวน node ใน subtree ด้านซ้ายมีไม่ถึง  node ให้ท่องต้นไม้ต่อไปทางด้านขวา แต่เราจะไม่ได้หาตัวที่  อีกต่อไป เนื่องจากเราข้ามทางด้านซ้ายและ root ไปแล้ว จึงต้องลบค่า  ด้วยจำนวนข้อมูลที่ข้ามไป

การเก็บ BST ในลักษณะดังกล่าวมีชื่อเรียกคือ Order statistics tree โดยสามารถศึกษาข้อมูลเพิ่มเติมได้ที่ <https://en.wikipedia.org/wiki/Order_statistic_tree> (ในลิงค์ อาจพิจารณาให้  เริ่มนับจาก  ถึง  ดังนั้นจึงตอบ root เมื่อ subtree ซ้ายมี  ตัวแทน)

หากใช้ Balanced Binary Search Tree จะทำให้แต่ละ operation ทำงานใน  เนื่องจากเราดำเนินการทั้งหมด  ครั้ง เวลารวมจึงเป็น 

2. Fenwick Tree หรือ Segment Tree

Fenwick Tree (Binary Indexed Tree) และ Segment Tree เป็นโครงสร้างข้อมูลที่รองรับการดำเนินการตอบคำถามเกี่ยวกับ subarray ได้อย่างรวดเร็ว อีกทั้งยังรองรับการแก้ไขสมาชิกใน array อีกด้วย โดยปกติแล้วจะใช้เวลา  ต่อ operation (โครงสร้างข้อมูลนี้เกินขอบเขตเนื้อหาระดับชาติ)

ในที่นี้ จะขอไม่กล่าวถึงวิธีการสร้างโครงสร้างข้อมูลนี้ขึ้นมา แต่จะนำโครงสร้างข้อมูลมาประยุกต์ใช้เลย ศึกษาข้อมูลเกี่ยวกับ Segment Tree ได้ที่ <https://www.geeksforgeeks.org/segment-tree-set-1-sum-of-given-range/> ส่วน Fenwick Tree (Binary Indexed Tree) ศึกษาได้ที่ <https://www.geeksforgeeks.org/binary-indexed-tree-or-fenwick-tree-2/>

เราจะใช้ Data Structure 1 ใน 2 ตัวนี้ รองรับ operation 2 อย่างคือ 1) หาว่าในช่วงตัวที่ 1 ถึง  มีผลรวมเท่ากับเท่าไหร่ และ 2) อัพเดทค่าของ array ช่องที่  โดยเราจะกำหนดให้ array ช่องที่  เป็น 1 เมื่อยังไม่ได้นำข้อมูลช่องที่  ออก และกำหนดเป็น 0 เมื่อนำข้อมูลช่องที่  ออกไปแล้ว

ยกตัวอย่าง หากมีข้อมูล  ตัว และนำข้อมูลตัวที่ 3, 6, 9, 10 ออกไปแล้ว จะได้ข้อมูลดังนี้

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ตำแหน่ง | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| ค่าที่เก็บ | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| ผลรวม | 1 | 2 | 2 | 3 | 4 | 4 | 5 | 6 | 6 | 6 | 8 | 9 |

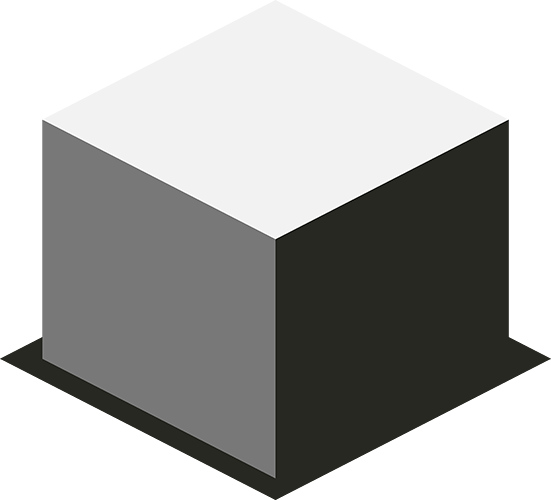
เมื่อเราต้องการหาตัวที่  ที่ยังไม่ถูกนำออก เราสามารถใช้การ Binary Search หาคำตอบได้ โดยเราจะหาตำแหน่งแรกที่มีผลรวมเท่ากับ  พอดี ในที่นี้ หาก  ก็จะต้องตอบตำแหน่งที่ 8 เพราะเป็นตำแหน่งแรกที่มีผลรวมเท่ากับ 6 พอดี

เนื่องจากเราต้อง Binary Search บนข้อมูล  ช่อง และการหาผลรวมแต่ละครั้งใช้เวลา  ดังนั้น จึงใช้เวลา  ต่อ operation ทำให้ Time Complexity ทั้งหมดเป็น  (สามารถลดเวลาให้เหลือ  ได้ โดย Binary Search บน Segment Tree ให้ทำวิธีคล้ายกับ Order Statistics Tree ข้างต้น)

(Solution Code อยู่ในหน้าถัดไป)

**Solution Code**

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  const int N = 1<<19;  int n, A[N], t[N+1];  void update(int x) {      for(int i = x; i <= N; i += i&-i) t[i]--;  }  int que(int x) {      int sum = 0;      for(int i = x; i != 0; i -= i&-i) sum += t[i];      return sum;  }  int query(int x) {      int l = 1, r = N;      while(l < r) {          int m = (l + r) >> 1;          if(que(m) >= x) r = m;          else l = m+1;      }      return l;  }  int main() {      for(int i = 1; i <= N; ++i) t[i] = i&-i;      scanf("%d", &n);      for(int i = 1; i <= n; ++i) scanf("%d", A+i);      for(int i = 1; i <= n; ++i) {          int now; scanf("%d", &now);          int ret = query(now);          printf("%d\n", A[ret]);          update(ret);      }  } |

 Worker by phon1209

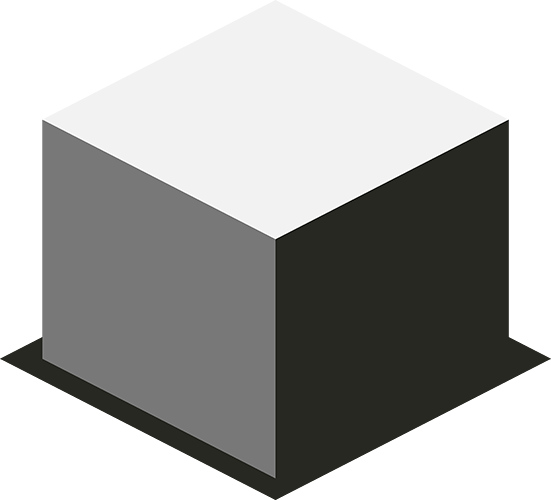
ในขณะที่โจทย์ข้อนี้พยายามแยกส่วนพนักงานกับเจ้าของบริษัทออกจากกันอย่างชัดเจน หากมองดูดี ๆ แล้วก็จะเห็นได้ว่าเราสามารถสมมุติให้เจ้าของบริษัทเป็นพนักงานอีกคนหนึ่ง โดยเจ้าของบริษัทเป็นคนแรกที่รู้งาน และสามารถส่งข่าวต่อให้ใครก็ได้ (ส่งให้ตัวแทนกลุ่มใดก็ได้) แล้วคนที่เหลือจะส่งต่อให้ใครอีกก็ได้ จนครบทุกคน

เราสามารถมองการส่งข่าวในลักษณะกราฟต้นไม้ได้ โดยที่มีเจ้าของบริษัทเป็นรากของต้นไม้ แล้วแตกกิ่งออกไปเรื่อย ๆ ตามลำดับ โดยเส้นเชื่อมแต่ละเส้นจะถูกกำ ากับด้วยน้ำหนัก ซึ่งก็คือเวลาที่ใช้ในการติดต่อสื่อสารนั่นเอง เนื่องจากว่ากราฟดังกล่าวจะต้องเชื่อมต่อคนทั้งกราฟ (ทุกคนต้องได้รับข่าว) โดยที่เวลารวมน้อยสุด โจทย์นี้ก็คือโจทย์ Minimum Spanning Tree ทั่วไปนั่นเอง

กราฟที่เราต้องการจะหา MST จะมี node ทั้งหมด  node แทนคนงานแต่ละคน (รวมเจ้าของบริษัทด้วย อาจจะแทนเจ้าของบริษัทด้วยเลข 0 หรือ n+1 แล้วแต่ความสะดวก) และเส้นเชื่อมระหว่างทุกคู่ node หากเป็นเส้นเชื่อมระหว่างเจ้าของบริษัทกับคนงานคนที่  น้าหนักจะเท่ากับ  หากเป็นเส้นเชื่อมระหว่างคนงานคนที่  กับคนที่  ก็จะมีน้ำหนักเท่ากับ  เมื่อสร้างกราฟเสร็จแล้ว ก็สามารถใช้ Prim’s Algorithm หรือ Kruskal’s Algorithm ได้เลย

**Solution Code**

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  using ll = long long;  using pii = pair<int, int>;  using edge = pair<int, pii>;  const int N = 1010;  int n, T[N], B[N], parent[N]  int root(int u)  {  if (parent[u] == u)  return u;  return parent[u] = root(parent[u]);  }  // continued on next page  bool merge(int u, int v)  {  u = root(u);  v = root(v);  if (u != v) {  parent[u] = v;  return true;  }  return false;  }  int main()  {  scanf("%d", &n);  vector<edge> E;  for (int i = 1; i <= n; ++i) {  scanf("%d", &T[i]);  E.emplace\_back(T[i], pii(0, i));  parent[i] = i;  }  for (int i = 1; i <= n; ++i) {  scanf("%d", &B[i]);  for (int j = 1; j < i; ++j)  E.emplace\_back(B[i]+B[j], pii(i, j));  }  sort(E.begin(), E.end());  ll sum = 0;  for (auto e : E) {  if (merge(e.second.first, e.second.second))  sum += e.first;  }  printf("%lld\n", sum);  return 0;  } |

 osu! mapping by AquaBlitz11

โจทย์ข้อนี้ถือเป็นโจทย์ข้อหนึ่งที่มีเนื้อเรื่องยาวที่สุดในบรรดาโจทย์ทั้งหมด ถึงอย่างไรก็ตาม หากพยายามอ่านก็จะทราบว่าโจทย์ข้อนี้เป็นโจทย์คลาสสิกที่ถูกดัดแปลงข้อหนึ่ง

สรุปสั้น ๆ เลยคือ กำหนด Weighted Directed Graph ให้ (กราฟถ่วงน้ำหนักแบบมีทิศทาง) ให้หาเส้นทางสั้นที่สุดจากจุด  ไปจุด  โดยมีเงื่อนไขพิเศษคือ จำนวน node ในเส้นทางจะต้องเป็นพหุคูณของค่า  ที่กำหนดให้

**Subtask 1-2 (50 คะแนน) -** 

ในกรณี  นั่นหมายความว่า เราต้องการหาเส้นทางสั้นสุดเฉย ๆ ไม่ได้มีเงื่อนไขพิเศษเพิ่มเติมอะไร เพราะฉะนั้นสามารถใช้ Dijkstra’s Algorithm ตามปกติได้เลย

**Subtask 3-4 (50 คะแนน) -** 

การใช้หมายเลข node ใน priority queue/distance array ไม่เพียงพอ เพราะจะทำให้เราสูญเสียข้อมูลจำนวน node ใน path ของเรา ดังนั้น แทนที่จะใช้หมายเลข node เพียงอย่างเดียว เราจะเก็บจำนวน node ที่เคยเดินผ่านมาใน path คู่กันไปด้วย เช่น หากเราต้องการพิจารณา node หมายเลข 5 โดยที่ node ปัจจุบันเป็น node ที่ 3 ใน path เราจะเก็บเป็นคู่อันดับ 

ในที่นี้ เราจะถือว่าคู่อันดับ  กับคู่อันดับ  เป็นคนละ node กันเลย ดังนั้น array ที่ใช้เก็บระยะห่างสั้นสุด จะต้องเป็น array 2 มิติ ถึงอย่างไรก็ตาม สังเกตว่าเราไม่จำเป็นต้องเก็บจำนวน node ที่เดินผ่านมาจริง ๆ ก็ได้ เก็บเพียงจำนวน node ใน mod  ก็เพียงพอ เพราะเราสนแค่ว่าจำนวน node สุดท้ายเป็นพหุคูณของ  หรือไม่ หากเป็นพหุคูณ ค่าที่เก็บใน mod  จะเท่ากับ 0

**คุณอาจจะใช้ Dijkstra’s Algorithm แบบผิด ๆ มาโดยตลอด**

หาก priority queue ของคุณเก็บแค่หมายเลข node (และจำนวน node ใน path) เพียงอย่างเดียว แล้วใช้ comparator function เพื่อจัดลำดับตาม distance array ด้านนอก โค้ดของคุณจะติด - อย่างน้อยหนึ่งตัว เนื่องจากเรา generate test case มาเพื่อดักวิธีนี้โดยเฉพาะ หลายคนอาจจะเข้าใจว่าใช้วิธีนี้ได้ แต่ความจริงแล้ว หากค่า distance ด้านนอกมีการเปลี่ยนแปลง อาจจะทำให้การจัดลำดับใน priority queue ผิดเพี้ยนได้ ดังนั้น เราควรเก็บ distance คู่กับหมายเลข node ไปเลย (เก็บเป็นคู่อันดับ) ส่วนการจัดลำดับ priority queue ให้ใช้ค่านี้เท่านั้น (ห้ามอ้างอิง array ด้านนอก) หากค่าด้านนอกเปลี่ยนแปลง ให้ push คู่อันดับใหม่ลงไป เมื่ออันเก่าถูก pop ออกมาแล้ว distance ไม่ตรงกับที่เก็บไว้ใน array ก็สามารถข้ามได้เลย

**Solution Code**

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  using ll = long long;  using pii = pair<int, int>;  using plii = pair<ll, pii>;  const ll INF = 1e17;  const int N = 10010;  const int M = 10010;  const int T = 10;  vector<pii> G[N];  bool visited[N][T];  ll dist[N][T];  int main()  {  int n, m, t, x, y, u, v, w;  scanf("%d%d%d%d%d", &n, &m, &t, &x, &y);  for (int i = 0; i < m; ++i) {  scanf("%d%d%d", &u, &v, &w);  G[u].emplace\_back(v, w);  }  for (int i = 1; i <= n; ++i) {  for (int j = 0; j < t; ++j)  dist[i][j] = INF;  }  dist[x][1%t] = 0;  priority\_queue<plii, vector<plii>, greater<plii> > Q;  Q.emplace(dist[x][1], pii(x, 1));  while (!Q.empty()) {  ll d = Q.top().first;  int u = Q.top().second.first;  int k = Q.top().second.second;  Q.pop();  if (visited[u][k]) continue;  visited[u][k] = true;  int nk = (k+1)%t;  for (auto v : G[u]) {  if (!visited[v.first][nk] && d+v.second < dist[v.first][nk]) {  dist[v.first][nk] = d+v.second;  Q.emplace(dist[v.first][nk], pii(v.first, nk));  }  }  }  printf("%lld\n", dist[y][0] == INF ? -1 : dist[y][0]);  return 0;  } |