מתרגל ממונה על התרגיל: סאג'י חשב, sajy@cs.technnion.ac.il

תאריך ושעת הגשה: 6/12/2022 בשעה 23:55

אופן ההגשה: בזוגות. אין להגיש ביחידים.(אלא באישור מתרגל אחראי של הקורס)

הנחיות כלליות:

- שאלות על התרגיל יש לפרסם באתר הפיאצה של הקורס תחת לשונית "wet_1":
  - האתר: piazza.com/technion.ac.il/winter2023/234218
  - נא לקרוא את השאלות של סטודנטים אחרים לפני שמפרסמים שאלה חדשה, למקרה שנשאלה כבר.
- נא לקרוא את המסמך "נהלי הקורס" באתר הקורס. בנוסף, נא לקרוא בעיון את כל ההנחיות בסוף מסמך זה.
- בפורום הפיאצה ינוהל FAQ ובמידת הצורך יועלו תיקונים כ**הודעות נעוצות** (Pinned Notes). תיקונים אלו **מחייבים**.
- התרגיל מורכב משני חלקים: יבש ורטוב.
  - לאחר קריאת כלל הדרישות, מומלץ לתכנן תחילה את מבני הנתונים על נייר. דבר זה יכול לחסוך לכם זמן רב.
  - לפני שאתם ניגשים לקודד את פתרונכם, ודאו כי יש לכם פתרון העומד בכל דרישות הסיבוכיות בתרגיל. תרגיל שאינו עומד בדרישות הסיבוכיות יחשב כפסול.
  - את הפתרון שלכם מומלץ לחלק למחלקות שונות שאפשר לממש (ולבדוק!) בהדרגתיות.
  - המלצות לפתרון התרגיל נמצאות באתר הקורס תחת: "Programming Tips Session".
    - המלצות לתכנות במסמך זה אינן מחייבות, אך מומלץ להיעזר בהן.
- העתקת תרגילי בית רטובים תיבדק באמצעות תוכנת בדיקות אוטומטית, המזהה דמיון בין כל העבודות הקיימות במערכת, גם כאלו משנים קודמות. לא ניתן לערער על החלטת התוכנה. התוכנה אינה מבדילה בין מקור להעתק! אנא הימנעו מהסתכלות בקוד שאינו שלכם.
- בקשות להגשה מאוחרת יש להפנות למתרגל האחראי בלבד בכתובת: barakgahtan@cs.technion.ac.il.

הקדמה:

כחלק מההכנות לגביע העולם בכדורגל (קטאר 2022), הארגון האחראי על ארגון המשחקים, הפיפ"א מעוניין ליצור אפליקציה שתאפשר למארגנים ומעריצים לעקוב אחרי התקדמות הטורניר.
הארגון מבקש את עזרתכם בבניית מבנה הנתונים שיהווה את הבסיס ל-backend של האפליקציה. בתמורה הוא מציע כרטיסים למונדיאל לכל זוג שעומד בכל הדרישות, וזוג כרטיסים למשחק הגמר עבור הזוג עם המימוש המהיר ביותר.
כל שחקן או קבוצה במערכת מיוצגים ע"י מזהה מספרי ייחודי. המערכת מאפשרת הכנסה והוצאה של שחקנים וקבוצות. המערכת מתחזקת סטטיסטיקות הקשורות לשחקנים, הקבוצות, ומאפשרת לסמלץ משחקים בין הקבוצות שתוצאותיהם נקבעת לפי הסטטיסטיקה השמורה במערכת.

דרוש מבנה נתונים למימוש הפעולות הבאות:

world_cup_t()

מאתחלת מבנה נתונים ריק. תחילה אין במערכת קבוצות או שחקנים.
פרמטרים: אין
ערך החזרה: אין
סיבוכיות זמן: $O(1)$ במקרה הגרוע.

---

virtual ~world_cup_t()

הפעולה משחררת את המבנה (כל הזיכרון אותו הקצאתם חייב להיות משוחרר).
פרמטרים: אין
ערך החזרה: אין
סיבוכיות זמן: $O(n+k)$ במקרה הגרוע, כאשר n הוא מספר השחקנים הכולל ו-k הוא מספר הקבוצות.

StatusType add_team(int teamId, int points)

הקבוצה בעלת מזהה teamId ייחודי משתתפת בתחרות, ולכן צריך להכניס אותה למבנה.
הניקוד ההתחלתי של הקבוצה הוא points.
בעת ההכנסה אין שחקנים בקבוצה.
פרמטרים:

teamId	מזהה הקבוצה החדשה.
points	הניקוד ההתחלתי של הקבוצה.

ערך החזרה:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם teamId<=0 או points<0.
FAILURE	אם teamId הוא מזהה של קבוצה קיימת.
SUCCESS	במקרה של הצלחה.

סיבוכיות זמן: $O(\log k)$ במקרה הגרוע, כאשר k הוא מספר הקבוצות במערכת.

StatusType remove_team(int teamId)

הקבוצה בעלת מזהה teamId נפסלת מהתחרות, ולכן צריך להוציאה מהמערכת.
אם קיימים שחקנים בקבוצה - לא ניתן למחוק אותה, והיא נשארת במערכת; והפעולה נכשלת.

פרמטרים:

teamId	מזהה הקבוצה.

ערך החזרה:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם teamId<=0.
FAILURE	אם אין קבוצה בעלת מזהה teamId או שהקבוצה אינה ריקה.
SUCCESS	במקרה של הצלחה.

סיבוכיות זמן: $O(\log k)$ במקרה הגרוע, כאשר k הוא מספר הקבוצות במערכת.

StatusType add_player(int playerId, int teamId, int gamesPlayed, int goals, int cards, bool goalKeeper)

השחקן בעל מזהה ייחודי playerId משחק עבור הקבוצה teamId בתחרות.
השחקן שיחק עד כה במספר gamesPlayed משחקים בהן הבקיע מספר - goals שערים, מספר הכרטיסים שקיבל (צהובים ואדומים יחד) הוא cards, ומשתנה ה- goalKeeper שלו קובע אם שחקן זה יכול לשחק כשוער או לא (בנוסף לכל שאר התפקידים).

פרמטרים:

playerId	מזהה השחקן שצריך להוסיף.
teamId	מזהה הקבוצה של השחקן.
gamesPlayed	מספר המשחקים ההתחלתי של השחקן.
goals	מספר השערים ההתחלתי שהשחקן הבקיע כבר.
cards	מספר הכרטיסים שהשחקן קיבל.
goalKeeper	האם השחקן יכול לשחק כשוער או לא.

ערך החזרה:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם playerId<=0, teamId<=0, gamesPlayed<0, goals<0 או cards<0. בנוסף, אם gamesPlayed=0 אך goals>0 או cards>0.
FAILURE	אם קיים כבר שחקן עם מזהה playerId או שהקבוצה עם המזהה teamId לא קיימת.
SUCCESS	במקרה של הצלחה.

סיבוכיות זמן: $O(\log n + \log k)$ במקרה הגרוע, כאשר n הוא מספר השחקנים במערכת, ו-k הוא מספר הקבוצות במערכת.

StatusType remove_player(int playerId)

השחקן בעל מזהה playerId נפצע במהלך התחרות, ולכן לא יוכל להמשיך להשתתף בתחרות, ולכן צריך להוציאו ממבנה הנתונים.

פרמטרים:

playerId	מזהה השחקן.

ערך החזרה:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם playerId<=0.
FAILURE	אם אין שחקן עם מזהה playerId.
SUCCESS	במקרה של הצלחה.

סיבוכיות זמן: $O(\log n)$ במקרה הגרוע, כאשר n הוא מספר השחקנים במערכת.

StatusType update_player_stats(int playerId, int gamesPlayed, int scoredGoals, int cardsReceived)

השחקן בעל מזהה playerId שיחק gamesPlayed משחקים (בנוסף למה ששיחק עד כה) ולכן יש צורך לעדכן את הסטטיסטיקות שלו.

במסגרת משחקים אלו (gamesPlayed), השחקן הבקיע scoredGoals שערים וקיבל cardsReceived כרטיסים.

הפרמטר gamesPlayed יכול לקבל את הערך 0, אז זה פשוט עדכון סטטיסטיקה עבור משחקים קודמים (כדוגמת אלו מ-play_match) ולא צריך להגדיל את מספר המשחקים ששחקן זה שיחק בהם.

<u>פרמטרים</u>:

playerId	מזהה השחקן.
gamesPlayed	מספר המשחקים עבורם יש עדכון.
scoredGoals	מספר השערים שהשחקן הבקיע.
cardsReceived	מספר הכרטיסים שהשחקן קיבל.

<u>ערך החזרה</u>:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם playerId<=0, gamesPlayed<0, scoredGoals<0 או cardsReceived<0.
FAILURE	אם אין שחקן עם מזהה playerId.
SUCCESS	במקרה של הצלחה.

<u>סיבוכיות זמן</u>: $O(\log n)$ במקרה הגרוע, כאשר n הוא מספר השחקנים במערכת.

StatusType play_match(int teamId1, int teamId2)

שתי הקבוצות בעלות המזהים teamId1 ו-teamId2 משחקות אחת מול השנייה בתחרות.

קבוצה יכולה לשחק רק אם היא מכילה לפחות 11 שחקנים כאשר לפחות אחד מהשחקנים הוא שוער, אחרת הפעולה נכשלת.

תוצאת המשחק נקבעת לפי הנוסחה, לכל קבוצה בנפרד. הקבוצה בעלת הערך המספרי הגדול יותר, תנצח:

$$points + \sum_{Players} (\text{goals} - cards)$$

כלומר, סכום הניקוד הנוכחי של הקבוצה עם סכום כל השערים שהשחקנים בה הבקיעו, פחות מספר הכרטיסים הכולל שהשחקנים קיבלו.

אם שני הערכים שווים, אז המשחק מסתיים בתיקו, וכל אחת מהקבוצות מקבלת תוספת $+1$ לניקוד שלה.

אחרת, הקבוצה המנצחת (רק) מקבלת תוספת $+3$ לניקוד, והשנייה 0 נקודות.

מספר המשחקים שכל אחד מהשחקנים בשתי הקבוצות שיחק (gamesPlayed) גדל ב-1 עבור כל השחקנים.

<u>פרמטרים</u>:

teamId1	מזהה הקבוצה הראשונה.
teamId2	מזהה הקבוצה השנייה.

<u>ערך החזרה</u>:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם teamId1<=0, teamId2<=0 או teamId2==teamId1.
FAILURE	אם אין קבוצה עם מזהה teamId1 או teamId2. או שאחת הקבוצות לא כשרה (11+שוער).
SUCCESS	במקרה של הצלחה.

<u>סיבוכיות זמן</u>: $O(\log k)$ במקרה הגרוע, כאשר k הוא מספר הקבוצות.

---

output_t < int > get_num_played_games(int playerId)

יש להחזיר את מספר המשחקים הכולל שהשחקן playerId השתתף בהם.

פרמטרים:

playerId מזהה השחקן.

ערך החזרה: מספר המשחקים הכולל בהם השתתף השחקן, ובנוסף סטטוס:

ALLOCATION_ERROR במקרה של בעיה בהקצאה/שחרור זיכרון.

INVALID_INPUT אם playerId <=0.

FAILURE אם אין שחקן עם מזהה playerId.

SUCCESS במקרה של הצלחה.

סיבוכיות זמן: $O(\log n)$ במקרה הגרוע, כאשר n הוא מספר השחקנים במערכת.

---

output_t < int > get_team_points(int teamId)

יש להחזיר את מספר הנקודות הנוכחי של הקבוצה בעלת המזהה teamId.

פרמטרים:

teamId מזהה הקבוצה.

ערך החזרה: מספר הנקודות הנוכחי של הקבוצה, ובנוסף סטטוס:

ALLOCATION_ERROR במקרה של בעיה בהקצאה/שחרור זיכרון.

INVALID_INPUT אם teamId<=0.

FAILURE אם אין קבוצה עם מזהה teamId.

SUCCESS במקרה של הצלחה.

סיבוכיות זמן: $O(\log k)$ במקרה הגרוע, כאשר k הוא מספר הקבוצות.

StatusType unite_teams(int teamId1, int teamId2, int newTeamId)

הקבוצות teamId1 ו-teamId2 מחליטות לאחד כוחות ולהשתתף תחת קבוצה אחת מאוחדת בעלת המזהה newTeamId. המזהה החדש יכול להיות אחד מ-teamId1 ו-teamId2 או מזהה חדש.

הקבוצה החדשה תכיל את כל השחקנים של שתי הקבוצות המקוריות, והניקוד שלה יהיה סכום הניקוד של שתיהן.

פרמטרים:

teamId1 מזהה קבוצה ראשונה.

teamId2 מזהה קבוצה שניה.

newTeamId מזהה קבוצת האיחוד.

ערך החזרה:

ALLOCATION_ERROR במקרה של בעיה בהקצאה/שחרור זיכרון.

INVALID_INPUT אם newTeamId <=0, teamId2<=0, teamId2==teamId1 או newTeamId<=0.

FAILURE אם אין קבוצות עם המזהה teamId2/teamId1 , או קיימת קבוצה עם המזהה newTeamId שהיא לא teamId2/teamId1.

SUCCESS במקרה של הצלחה.

סיבוכיות זמן: $O(\log k + n_{Team1ID} + n_{Team2ID})$ במקרה הגרוע, כאשר k הוא מספר הקבוצות.

$n_{Team1ID}$ ו- $n_{Team2ID}$ הם מספרי השחקנים בכל אחת מהקבוצות המקוריות.

---

output_t < int > get_top_scorer(int teamId)

כדי לספק ממשק לאפליקציית פנטזי, רוצים לדעת מי השחקן שהבקיע הכי הרבה שערים בקבוצה מסוימת, או מבין כל השחקנים במערכת.

אם teamId>0, הפעולה תחזיר את השחקן שהבקיע הכי הרבה שערים בקבוצה בעלת מזהה teamId.

אם teamId<0, הפעולה תחזיר את השחקן שהבקיע הכי הרבה שערים מבין כל השחקנים במערכת.

בשני המקרים, אם קיים יותר משחקן עם אותו מספר שערים, אז מוחזר השחקן עם מספר הכרטיסים הנמוך יותר.

אם יש יותר משחקן אחד מתאים, יוחזר השחקו בעל המזהה הגבוה ביותר.

פרמטרים:

teamId	מזהה הקבוצה.

ערך החזרה: המזהה של השחקן שהבקיע הכי הרבה שערים, ובנוסף סטטוס:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם teamId==0.
FAILURE	אם teamId>0 ואין קבוצה עם מזהה זה, או אם teamId>0 ואין שחקנים בקבוצה עם מזהה זה או אם teamId<0 ואין שחקנים במערכת
SUCCESS	במקרה של הצלחה.

סיבוכיות זמן: אם teamId<0, אז $O(1)$ במקרה הגרוע. אחרת, $O(\log k)$ במקרה הגרוע, כאשר k הוא מספר הקבוצות.

---

output_t < int > get_all_players_count(int teamId)

אם teamId>0, הפעולה תחזיר את מספר השחקנים בקבוצה teamId.

אם teamId<0, הפעולה תחזיר את מספר השחקנים במערכת.

פרמטרים:

teamId	מזהה הקבוצה.

ערך החזרה: מספר השחקנים בקבוצה או בכל המערכת, ובנוסף סטטוס:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם teamId==0.
FAILURE	אם teamId>0 ואין קבוצה עם מזהה זה.
SUCCESS	במקרה של הצלחה.

סיבוכיות זמן:

אם teamId<0, אז $O(1)$ במקרה הגרוע,

אחרת $O(\log k)$ במקרה הגרוע, כאשר k הוא מספר הקבוצות.

StatusType get_all_players(int teamId, int * const output)

כדי לפרסם דירוג יומי של כל השחקנים בתחרות, רוצים פונקציה שמחזירה מערך של כל השחקנים במערכת, או בקבוצה מסוימת.

אם teamId>0, הפעולה ממלאת את מערך הפלט של כל השחקנים בקבוצה teamId.

אם teamId<0, הפעולה ממלאת את מערך הפלט של כל השחקנים במערכת.

בשני המקרים, המזהים של השחקנים יהיו ממוינים בסדר עולה לפי מספר השערים שכל שחקן הבקיע.

במקרה של שוויון, השחקן בעל מספר הכרטיסים הגבוה יותר יופיע לפני.

במקרה של שוויון נוסף, השחקן בעל המזהה הגדול יותר יופיע אחרי.

output מצביע למקום הראשון במערך הפלט. המערך מוקצה (בגודל המתאים) ומשוחרר בקוד שקורא לפונקציה זו.

הגודל המתאים זה הערך המוחזר מ-get_all_players_count עבור אותו הקלט, במקרה של הצלחה.

פרמטרים:

teamId	מזהה הקבוצה הרצויה.
output	מערך המזהים המוחזר במקרה הצלחה.

ערך החזרה:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם output==NULL או teamId==0.

FAILURE אם teamId>0 ואין קבוצה עם מזהה זה, או אם teamId<0 ואין שחקנים בקבוצה עם מזהה זה או אם teamId<0 ואין שחקנים במערכת.

SUCCESS במקרה של הצלחה.

<u>סיבוכיות זמן:</u> אם teamId<0, אז $O(n)$ במקרה הגרוע, כאשר n הוא מספר השחקנים במערכת.
אחרת, $O(\log k + n_{\text{TeamID}})$ במקרה הגרוע, כאשר k הוא מספר הקבוצות ו- n_{TeamID} זה מספר השחקנים בקבוצה בעלת המזהה הרלוונטי.

---

output_t < int > get_closest_player(int playerId, int teamId)

לצד התחרות הרגילה, מארגני התחרות מעוניינים לקיים תחרות בין זוגות של שחקנים, אחד-על-אחד.
על מנת לקבל תחרות הוגנת, נרצה שכל אחד ישחק מול השחקן עם הסטטיסטיקות שהכי קרובות אליו.
עבור השחקן בעל מזהה playerId שמשחק בקבוצה בעלת מזהה teamId הפונקציה תחזיר את השחקן שהכי "קרוב אליו" מבין כל שאר השחקנים במערכת ברגע זה.
הכוונה בשחקנים קרובים אחד לשני היא שהם עוקבים בדירוג השחקנים - הסדר של השחקנים שמוגדר בפונקציה get_all_players. במידה והשחקן הוא לא הראשון או האחרון בדירוג, במצב זה יש לו 2 אפשרויות לשחקן קרוב ביותר, (זה שלפניו או זה שאחריו) ולכן לאחר מכן נפעיל את כללי השוויון:
השחקן אשר יוחזר הוא זה עם מספר השערים הקרוב ביותר. (מבין אותם 2)
במקרה של שוויון, מוחזר זה עם מספר הכרטיסים הקרוב ביותר. (מבין אותם 2)
במקרה של שוויון נוסף, מוחזר זה עם המזהה הקרוב ביותר.(מבין אותם 2)
לבסוף, אם עדיין יש שוויון, מוחזר השחקן עם המזהה הגדול יותר.(מבין אותם 2)

אם השחקן הוא הראשון, אז השחקן הקרוב ביותר אליו הוא השני.
כדומה, אם הוא האחרון, אז שחקן לפני אחרון הוא הכי קרוב אליו.

דוגמא מפורטת:
נייצג כל שחקן עם השלשה: <goals, cards, playerId>.
נניח שיש לנו את השחקנים הללו במערכת, ליד כל שלשה רשום ה-closest_player שלו:

<1, 20, 21> -> 25
<1, 20, 25> -> 21
<1, 5, 3> -> 4
<1, 1, 4> -> 3
<2, 5, 5> -> 6
<2, 4, 6> -> 5
<2, 3, 24> -> 7
<2, 2, 7> -> 8
<2, 1, 8> -> 7
<5, 30, 9> -> 10
<5, 29, 10> -> 11
<5, 28, 11> -> 12
<5, 27, 12> -> 11
<6, 40, 51> -> 52
<6, 40, 52> -> 53
<6, 40, 53> -> 52
<6, 10, 54> -> 60
<6, 5, 60> -> 70
<6, 4, 70> -> 60
<8, 6, 100> -> 80
<10, 7, 80> -> 90
<10, 7, 90> -> 80

תוודאו שאתם יודעים למה כל שורה נכונה.

<u>פרמטרים</u>:

playerId	מזהה השחקן.
teamId	מזהה הקבוצה של השחקן.

<u>ערך החזרה</u>: מזהה השחקן שהכי דומה לשחקן הקלט, ובנוסף סטטוס:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם playerId<=0 או teamId<=0.
FAILURE	אם השחקן עם מזהה playerId אינו מזהה של שחקן שמשחק בקבוצה teamId, או שזה השחקן היחיד בכל מערכת.
SUCCESS	במקרה של הצלחה.

<u>סיבוכיות זמן</u>: $O(\log k + \log n_{TeamID})$ במקרה הגרוע, כאשר k הוא מספר הקבוצות ו- n_{TeamID} זה מספר השחקנים בקבוצה בעלת המזהה הנ"ל.

---

output_t < int > knockout_winner(int minTeamId, int maxTeamId)

בניסיון לחזות מי הקבוצה שהולכת לנצח בגביע העולם, רוצים פונקציה שמדמה תהליך נוקאאוט בין חלק מהקבוצות במערכת.

מבין כל הקבוצות שהמזהה שלהן מקיים $\text{minTeamId} \le \text{teamId} \le \text{maxTeamId}$, אלו שמכילות לפחות 11 שחקנים וגם לפחות אחד מהשחקנים הוא שוער, מתחרות אחת עם השנייה.

התחרות מתקיימת באופן הבא: מסתכלים על כל הקבוצות הכשרות (מקיימות את שני התנאים), מסודרות לפי מזהה הקבוצה בסדר עולה.

הקבוצה הראשונה משחקת עם השנייה, השלישית עם הרביעית, וכך הלאה. ייתכן שנשארת קבוצה אחת שלא משחקת עם אף קבוצה אחרת בסיבוב זה. הקבוצה המנצחת בכל משחק נקבעת לפי הנוסחה בפונקציה $play_match$. במקרה של שוויון, הקבוצה עם המזהה הגדול יותר מנצחת. הקבוצה שמנצחת מקבלת 3+ נקודות, ובנוסף מתאחדת עם הקבוצה המפסידה כמו ב-$unite_teams$ ושומרת על המזהה שלה.

ממשיכים בסבבים עד שמתקבלת קבוצה מנצחת יחידה שמוחזרת.

הפעולה <u>אינה</u> משנה את מבנה הנתונים, אלא רק מחזירה את מזהה הקבוצה המנצחת אם התחרות הייתה מתקיימת לפי המצב הנוכחי.

<u>דוגמה</u>:

נניח שיש את הקבוצות: $\{\langle 1,20\rangle, \langle 3,20\rangle, \langle 6,5\rangle, \langle 7,30\rangle, \langle 10,11\rangle\}$ כאשר המספר השמאלי בזוג הוא מזהה הקבוצה והימני הוא הערך שמוגדר ב-$play_match$ ולפיו נקבע מי מנצח.

אחרי סיבוב ראשון, מתקבל: $\{\langle 3,43\rangle, \langle 7,38\rangle, \langle 10,11\rangle\}$.

אחרי סיבוב שני, מתקבל: $\{\langle 3,84\rangle, \langle 10,11\rangle\}$.

בסיבוב האחרון, קבוצה עם מזהה 3 מנצחת ולכן זה הערך המוחזר.

<u>פרמטרים</u>:

minTeamId	המזהה המינימלי של קבוצה שהולכת לשחק.
maxTeamId	המזהה המקסימלי של קבוצה שהולכת לשחק.

<u>ערך החזרה</u>: מזהה הקבוצה המנצחת בתחרות, ובנוסף סטטוס:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם minTeamId<0, maxTeamId<0, maxTeamId<minTeamId.
FAILURE	אם אין אף קבוצה שיכולה לשחק (מזהה בתחום ולפחות 11 שחקנים שאחד לפחות שוער).
SUCCESS	במקרה של הצלחה.

<u>סיבוכיות זמן</u>: $O(\log(\min\{n,k\}) + r)$ במקרה הגרוע, כאשר k הוא מספר הקבוצות ו- n זה מספר השחקנים במערכת. r זה מספר הקבוצות המתחרות, כלומר, אלו שהמזהה שלהן בתחום המתאים ומכילות ולפחות 11 שחקנים שאחד לפחות שוער.

סיבוכיות מקום:

סיבוכיות המקום הדרושה עבור מבנה הנתונים היא $O(n+k)$ במקרה הגרוע, כאשר n הוא מספר השחקנים ו-k הוא מספר הקבוצות. כלומר בכל רגע בזמן הריצה, צריכת המקום של מבנה הנתונים תהיה לינארית בסכום מספרי השחקנים והקבוצות במערכת.

סיבוכיות המקום הנדרשת עבור כל פעולה (כלומר, זיכרון "העזר" שכל פעולה משתמשת בו) אינה מצוינת לכל פעולה לחוד, אך אסור לעבור את סיבוכיות המקום הדרושה שמוגדרת לכל המבנה.

ערכי החזרה של הפונקציות:

כל אחת מהפונקציות מחזירה ערך מטיפוס StatusType שייקבע לפי הכלל הבא:

- תחילה, יוחזר INVALID_INPUT אם הקלט אינו תקין.
- אם לא הוחזר INVALID_INPUT:
  - בכל שלב בפונקציה, אם קרתה שגיאת הקצאה/שחרור יש להחזיר ALLOCATION_ERROR. מצב זה אינו צפוי אלא באחד משני מקרים (לרוב): באמת השתמשתם בקלט גדול מאוד ולכן המבנה ניצל את כל הזיכרון במערכת, או שיש זליגת זיכרון בקוד.
  - אם קרתה שגיאה אחרת, כפי שמצוין בכל פונקציה, יש להחזיר מיד FAILURE מבלי לשנות את מבנה הנתונים.
- אחרת, יוחזר SUCCESS.

חלק מהפונקציות צריכות להחזיר בנוסף עוד פרמטר (לרוב int), לכן הן מחזירות אובייקט מטיפוס output_t<T>.

אובייקט זה מכיל שני שדות: הסטטוס (__status) ושדה נוסף (__ans) מסוג T.

במקרה של הצלחה (SUCCESS), השדה הנוסף יכיל את ערך החזרה, והסטטוס יכיל את SUCCESS. בכל מקרה אחר, הסטטוס יכיל את סוג השגיאה והשדה הנוסף לא מעניין.

שני הטיפוסים (output_t<T>,StatusType) ממומשים כבר בקובץ "wet1util.h" שניתן לכם כחלק מהתרגיל.

הנחיות:

חלק יבש:

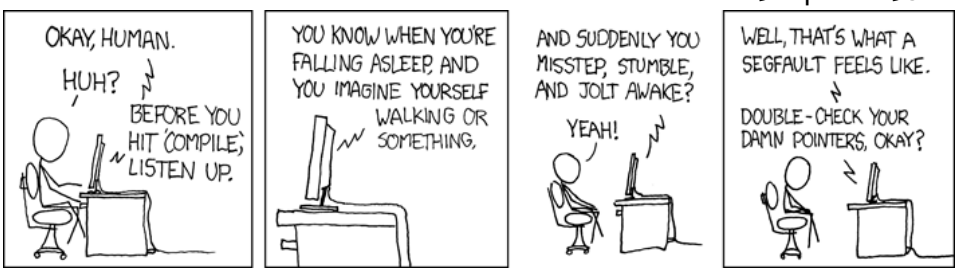
- החלק היבש הווה חלק מהציון על התרגיל כפי שמצוין בנהלי הקורס.
- לפני מימוש הפעולות בקוד יש לתכנן היטב את מבני הנתונים והאלגוריתמים ולוודא כי באפשרותכם לממש את הפעולות בדרישות הזמן והזיכרון שלעיל.
- הגשת החלק הרטוב מהווה תנאי הכרחי לקבלת ציון על החלק היבש, כלומר, הגשה בה יתקבל אך ורק חלק יבש תגרור ציון 0 על התרגיל כולו.
- יש להכין מסמך הכולל תיאור של מבני הנתונים והאלגוריתמים בהם השתמשתם בצירוף הוכחת סיבוכיות הזמן והמקום שלהם. חלק זה עומד בפני עצמו וצריך להיות מובן לקורא גם לפני העיון בקוד. אין צורך לתאר את הקוד ברמת המשתנים, הפונקציות והמחלקות, אלא ברמה העקרונית. חלק יבש זה לא תיעוד קוד.
- ראשית הציגו את מבני הנתונים בהם השתמשתם. רצוי ומומלץ להיעזר בציור.
- לאחר מכן הסבירו כיצד מימשתם כל אחת מהפעולות הנדרשות. הוכיחו את דרישות סיבוכיות הזמן של כל פעולה תוך כדי התייחסות לשינויים שהפעולות גורמות במבני הנתונים.
- הוכיחו שמבנה הנתונים וכל הפעולות עומדים בדרישת סיבוכיות המקום.
- החסמים הנתונים בתרגיל הם לא בהכרח הדוקים ולכן יכול להיות שקיים פתרון בסיבוכיות טובה יותר. מספיק להוכיח את החסמים הדרושים בתרגיל.
- רמת פירוט: יש להסביר את כל הפרטים שאינם טריוויאליים ושחשובים לצורך מימוש הפעולות ועמידה בדרישות הסיבוכיות. אין לדון בפרטים טריוויאליים (הפעילו את שיקול דעתכם בקשר לזה, ושאלו את האחראי על התרגיל אם אינכם בטוחים). אין לצטט קטעים מהקוד כתחליף להסבר. אין צורך לפרט אלגוריתמים שנלמדו בכתה. כמו כן, אין צורך להוכיח תוצאות ידועות שנלמדו בכתה, אלא מספיק לציין בבירור לאיזו תוצאה אתם מתכוונים.
- **על חלק זה לא לחרוג מ-8 עמודים.**
- והכי חשוב **keep it simple!**

חלק רטוב:

- מומלץ לממש תחילה את מבני הנתונים בצורה הכללית ביותר ורק אז לממש את הפונקציות הנדרשות בתרגיל.
- אנו ממליצים בחום על מימוש **Object Oriented**, ב**++C**, מימוש כזה יאפשר לכם להגיע לפתרון פשוט וקצר יותר לפונקציות אותן עליכם לממש ויאפשר לכם להכליל בקלות את מבני הנתונים שלכם (זכרו שיש תרגיל רטוב נוסף בהמשך הסמסטר).
- על הקוד להתקמפל על csl3 באופן הבא:

g++ -std=c++11 -DNDEBUG -Wall *.cpp

עליכם מוטלת האחריות לוודא קומפילציה של התכנית ב+ + g. אם בחרתם לעבוד בקומפיילר אחר, מומלץ לקמפל ב+ + g מידי פעם במהלך העבודה.



הערות נוספות:

- חתימות הפונקציות שעליכם לממש ומספר הגדרות נמצאים בקובץ worldcup23a1.h.
- קראו היטב את הקובץ הנ"ל, לפני תחילת העבודה.
- אין לשנות את הקבצים main23a1.cpp ו-wet1util.h אשר סופקו כחלק מהתרגיל, **ואין להגיש אותם**.
  - את שאר הקבצים ניתן לשנות.
  - תוכלו להוסיף קבצים נוספים כרצונכם, ולהגיש אותם.
  - העיקר הוא שהקוד שאתם מגישים יתקמפל עם הפקודה לעיל, כאשר מוסיפים לו את שני הקבצים main23a1.cpp ו-wet1util.h.
- עליכם לממש בעצמכם את כל מבני הנתונים (למשל **אין להשתמש במבנים של STL** ואין להוריד מבני נתונים מהאינטרנט). **כחלק מתהליך הבדיקה אנו נבצע בדיקה ידנית של הקוד ונוודא שאכן מימשתם את מבני הנתונים שבהם השתמשתם.**
- בפרט, אסור להשתמש ב-std::iterator, std::vector, std::pair, או כל אלגוריתם של STL.

- **ניתן** להשתמש במצביעים חכמים (Smart pointers כמו shared_ptr), בספריית math או בספריית exception.
- חשוב לוודא שאתם מקצים/משחררים זיכרון בצורה נכונה (מומלץ לוודא עם valgrind). לא חייבים לעבוד עם מצביעים חכמים, אך אם אתם מחליטים כן לעשות זאת, לוודא שאתם משתמשים בהם נכון. (תזכרו שהם לא פתרון קסם, למשל, כאשר יוצרים מעגל בהצבעות)
- שגיאות של ALLOCATION_ERROR בד"כ מעידות על זליגה בזיכרון.
- מצורפים לתרגיל קבצי קלט ופלט לדוגמא, ניתן להריץ את התוכנה על הקלט ולהשוות עם הפלט המצורף.
- **שימו לב:** התוכנית שלכם תיבדק על קלטים שונים מקבצי הדוגמא הנ"ל, שיהיו ארוכים ויכללו מקרי קצה שונים. לכן, מומלץ **מאוד** לייצר בעצמכם קבצי קלט, לבדוק את התוכנית עליהם, ולוודא שהיא מטפלת נכון בכל מקרה הקצה.

הגשה:

- **חלק יבש+ חלק רטוב**:

הגשת התרגיל הנה **אך ורק** אלקטרונית דרך אתר הקורס.
יש להגיש קובץ **ZIP** שמכיל את הדברים הבאים:

- בתיקייה הראשית:
  - קבצי ה-Source Files שלכם. **למעט הקבצים main23a1.cpp ו-wet1util.h**, שאסור לשנות.
  - קובץ PDF בשם dry.pdf אשר מכיל את הפתרון היבש. מומלץ להקליד את החלק הזה אך ניתן להגיש קובץ PDF מבוסס על סריקה של פתרון כתוב בכתב יד. שימו לב כי במקרה של כתב לא קריא, כל החלק השני לא תיבדק.
  - קובץ submissions.txt, המכיל בשורה הראשונה את שם, תעודת הזהות וכתובת הדוא"ל של השותף הראשון ובשורה השנייה את שם, תעודת הזהות וכתובת הדוא"ל של השותף השני. לדוגמה:

John Doe 012345678 doe@cs.technion.ac.il
Henry Taub 123456789 taub@cs.technion.ac.il

- **שימו לב כי אתם מגישים את כל שלושת החלקים הנ"ל.**
- אין להשתמש בפורמט כיווץ אחר (לדוגמה RAR), מאחר ומערך הבדיקה האוטומטי אינו יודע לזהות פורמטים אחרים.
- **יש לוודא שכאשר נכנסים לקובץ הזיפ הקבצים מופיעים מיד בתוכו ולא בתוך תיקיה שבתוך קובץ הזיפ.** עבור הגשה שבה הקבצים יהיו בתוך תיקייה, הבדיקה האוטומטית לא תמצא את הקבצים ולא תוכל לקמפל ולהריץ את הקוד שלכם ולכן תיתן אוטומטית 0.
- לאחר שהגשתם, יש באפשרותכם לשנות את התוכנית ולהגיש שוב. ההגשה האחרונה היא הנחשבת.
- הגשה שלא תעמוד בקריטריונים הנ"ל תפסל ותקנס בנקודות!
  - אחרי שאתם מכינים את ההגשה בקובץ zip מומלץ מאוד לקחת אותה לשרת ולהריץ את הבדיקות שלכם עליה כדי לוודא שאתם מגישים את הקוד שהתכוונתם להגיש בדיוק (ושהוא מתקמפל).

דחיות ואיחורים בהגשה:

- דחיות בתרגיל הבית תינתנה אך ורק לפי תקנון הקורס.
- 5 נקודות יורדו על כל יום איחור בהגשה ללא אישור מראש. באפשרותכם להגיש תרגיל באיחור של עד 5 ימים ללא אישור. תרגיל שיוגש באיחור של יותר מ-5 ימים ללא אישור מראש יקבל 0.
- במקרה של איחור בהגשת התרגיל יש עדיין להגיש את התרגיל אלקטרונית דרך אתר הקורס.
- בקשות להגשה מאוחרת יש להפנות למתרגל האחראי בלבד בכתובת barakgahtan@cs.technion.ac.il. לאחר קבלת אישור במייל על הבקשה, מספר הימים שאושרו לכם נשמר אצלנו. לכן, אין צורך לצרף להגשת התרגיל אישורים נוספים או את שער ההגשה באיחור.

בהצלחה!