**תכנות מקבילי ומבוזר – תרגיל בית 3**

**מגישים:**

מיכאל תורג'מן, 307910984, [michaeltur3@gmail.com](mailto:michaeltur3@gmail.com)

נדב לוינטוב, 203906425, [dsp155@gmail.com](mailto:dsp155@gmail.com)

**חלק א' – tsp\_static**

בחלק זה מימשנו גרסה סטאטית של אלגוריתם Branch and Bound מבוזר, בגרסה זו תהליך בעל דרגה 0 מקבל את כל ה-input ומחלק את העבודה לכל התהליכים האחרים. בסיום העבודה כל התהליכים מעבירים את התוצאות שלהם לתהליך בעל דרגה 0 וזה בוחר את התוצאה הטובה ביותר.

החלוקה מתבצעת מראש ולא מתחשבים בכמות העבודה בפועל, כלומר חלק מהתהליכים יבצעו יותר עבודה מאחרים – זאת משום שאלגוריתם ה-Branch and Bound מבצע גיזום של ענפים בעץ החיפוש במידה כאשר ברור שסיור בענף זה אינו יוביל לתוצאה יותר טובה מהתוצאה שכבר מצאנו.

חישוב lower bound עבור אלגוריתם branch and bound:

(זאת מכיוון שסכום המעגל חסום מלמטה ע"י חצי מסכום 2 הקשתות המינימליות של כל עיר, כופלים בחצי משום שחישבנו כל קשת פעמיים)

שלבי הפתרון:

1. תהליך 0 מקבל את הנתונים, מחשב את כמות העבודה שיש לחלק על בסיס מספר התהליכים הזמינים לו (חישוב כמות הרישות שיש ליצור מתבצע ע"י חישוב אורך הרישא המינימלי שעבורו יש עבודה לכל התהליכים), יוצר רישות של מסלול לכל תהליך ושולח לכל תהליך את המידע הנחוץ לו :
   1. ראשית נשלח את גודל המסלול שיש למצוא (מספר הערים) – מידע זה נשלח קודם משום שיש להשתמש בו בכדי להכין את גודל המערכים אשר יכילו את שאר המידע.
   2. מערכי הקואורדינטות של הערים והרישא המתאימה למסלול של התהליך – זו שליחה שניה לכל תהליך כאשר כל תהליך מגדיר struct חדש של מידע אשר מכיל את המידע (לכל תהליך נשלח את הרישא הראשונה ואת מספר הרישות, והוא יחשב לבד מתוך הרישא הראשונה את הרישות הבאות).

השליחה מתבצעת בעזרת פונקציית MPI\_Bsend.

1. לאחר שכל התהליכים קבלו את הרישות שלהם מתהליך ה-0 (נוודא זאת על ידי barrier) כל תהליך (כולל תהליך 0) מבצע Branch and Bound לפי הרישא שקיבל מתהליך ה-0 כאשר בסיום האלגוריתם לכל תהליך יש מסלול קצר ביותר שמתחיל ברישא אותה קיבל מתהליך 0. נשים לב שיתכן שתהליך יקבל יותר מרישא אחת התחלתית וכל תהליך יבדוק את כל המסלולים עבור כל הרישות שקיבל, בנוסף יתכנו תהליכים שיקבלו יותר רישות לבדוק, זאת בהתאם לגודל המסלול הנדרש וכן כמות התהליכים הזמינה אך נשים לב שההבדל הוא לכל היותר של רישא אחת – כלומר לא יכול להיות שיהיה תהליך שיידרש לבצע Branch and Bound על שתי רישות יותר מאשר תהליך אחר.
2. תהליך ה-0 מבצע Gather של תוצאות האלגוריתם של כל התהליכים (כולל של עצמו, למעשה כל תהליך מבצע את פונקציית MPI\_Gather כשהיעד הוא תהליך 0) כלומר, מכל תהליך נקבל מסלול קצר ביותר המתחיל ברישא אותה קיבל מתהליך ה-0 וכן את משקל המסלול. תהליך ה-0 ימצא מבין כל התוצאות את המסלול בעל הערך המינימלי (או אחד מהם אם יש כמה מסלולים בעלי אותו ערך מינימלי).

**חלק ב' – tsp**

בחלק זה מימשנו גרסה דינאמית של אלגוריתם Branch and Bound מבוזר, בגרסה זו תהליך בעל דרגה 0, שנקרא master, מחלק עבודות קטנות לשאר התהליכים, שנקראים workers.

(בחלק זה אנו משתמשים באותו אלגוריתם branch and bound ובאותו lower bound)

שלבי הפתרון:

תהליך ה-master -

1. בדומה לחלק הקודם, תהליך 0 מקבל את הנתונים, מחשב את כמות העבודה שיש לחלק על בסיס מספר התהליכים הזמינים לו, יוצר רישות של מסלול לכל תהליך ושולח לכל תהליך את המידע הנחוץ לו. בניגוד לסעיף הקודם יש 2 שינויים:
   1. בחלק זה נתון לנו כי לכל התהליכים נתון הקלט, לכן אין צורך לשלוח להם את מערכי הקואורדינטות. לכן ניצור struct חדש של job שיכלול בתוכו את הרישא שיש לחשב, את האורך שלה, והאם זה job שיש לבצע (זאת מכיוון שיכול לקרות מצב שבו מראש אין מספיק jobs לכל התהליכים – למשל מספר ערים קטן והרבה תהליכים, אבל אנו בכל מקרה צריכים לשלוח job ראשוני לכל התהליכים בגלל אופן השליחה, יוסבר בהמשך).
   2. מכיוון שבחלק הדינאמי אנו רוצים ליצור הרבה עבודות קטנות יותר, בחלק זה ניצור רישות ארוכות יותר לעומת החישוב המתבצע בחלק הקודם. החישוב מתבצע באמצעות חישוב כמות הרישות מהחלק הקודם, אך כעת אנו מתנהגים כאילו יש לנו הרבה יותר תהליכים. לכן, השתמשנו בפונקציה שחישבה את כמות הרישות מהחלק הקודם, ושלחנו לה את מספר התהליכים מוכפל בפקטור כלשהו. (ממספר הרצות מבחן ראינו כי אנו מקבלים את השיפור הטוב יותר עבור פקטור 10,000).

בהתחלה כל ה-workers ממתינים ל-job, לכן אנו שולחים את העבודה הראשונה באמצעות MPI\_Scatter (ה-master מקבל job דמה).

1. לאחר מכן, תהליך ה-master ממתין בלולאה לתוצאות ביניים שהוא מקבל מה-workers. במידה והוא מקבל תוצאה טובה יותר ממה שהוא מכיר, הוא שומר את התוצאה הזאת אצלו.

תוצאות הביניים נשלחות ב-struct תוצאה שאנו יוצרים שכולל בתוכו מסלול ואת העלות שלו.

1. לאחר קבלת תוצאת ביניים, תהליך ה-master שולח את העבודה הבאה לביצוע לתהליך שממנו הוא קיבל את ההודעה. עבור שליחה זו השתמשנו בפקודה MPI\_Ssend. הפקודה אמורה לחזור יחסית מהר, משום שה-worker ממתין לעבודה נוספת מיד איך שהוא שולח את התשובה המיידית.
2. כאשר נגמרות לתהליך ה-master כל העבודות לשליחה, הוא ממתין כעת לקבל את התוצאות החלקיות האחרונות מכל התהליכים הפעילים.
3. כאשר כל התהליכים סיימו את העבודה שלהם, תהליך ה-master שולח להם הודעה מיוחדת המסמנת להם לסיים. ההודעה היא job שמסומן ב-tag המסמן לתהליך שיש לסיים.
4. תהליך ה-master מחזיר את התוצאה הטובה ביותר שקיבל.

תהליך ה-worker -

1. ראשית worker מקבל את העבודה הראשונה שלו באמצעות הפקודה MPI\_Scatter.
2. worker מחשב את התוצאה החלקית שלו באותו אופן כמו בחלק א' עבור הרישא שהוא קיבל.
3. כעת, התהליך בודק האם קיים worker אחר שמצא פתרון טוב יותר מהפתרון הכי טוב שמוכר לכולם. הוא מבצע זאת באמצעות הפקודה MPI\_Iprobe על tag ספציפי המציין שליחה של תוצאה טובה יותר. אם יש הודעה כזאת הוא מקבל אותה באמצעות MPI\_Recv, ומעדכן את התוצאה הזאת אצלו. (משתמשים בפקודה לא חוסמת, כי לא בטוח שיש הודעה)
4. במידה והתהליך מצא תוצאה טובה יותר מהתוצאה הכי טובה, הוא יעדכן את כל שאר התהליכים עליה. עדכון זה הוא יבצע באמצעות MPI\_Bsend בלולאה לכל תהליכי ה-workers של הודעה הכוללת struct שאנו בונים שכולל את המחיר בלבד. (כמות ה-buffers שמשתמשים בהם לצורך התקשורת תלויה במספר התהליכים, אך לפי התרגיל מותר).

כאן אנחנו רוצים לחזור כמה שיותר מהר כדי להמשיך לשלוח לאחר מכן להמשיך לעבוד לכן בחרנו בפקודה זו.

1. במידה והתהליך מצא פתרון טוב יותר מהפתרון הידוע, הוא ישלח אותו לתהליך ה-master. אחרת, הוא ישלח "פתרון" דמה שהמשקל שלו הוא INT\_MAX ואז תהליך ה-master יתעלם ממנו. את השליחה הוא יבצע באמצעות הפקודה MPI\_Ssend (במילא הוא צריך לחכות לפקודה נוספת מה-master, ובמידה ותהיה לו עבודה חדשה הוא בכל מקרה יבדוק שוב האם קיימת תוצאה טובה יותר. לכן בחרנו בפקודה זו)
2. לאחר שליחת הפתרון, התהליך ממתין לקבל הודעה. הוא יכול לקבל אחת מ-3 ההודעות הבאות:
   1. עבודה חדשה – במקרה כזה הוא יבצע אותה כמו קודם.
   2. תוצאה טובה יותר חדשה – במקרה כזה הוא יעדכן את התוצאה הטובה ביותר שהוא מכיר ויחזור לסעיף 6.
   3. הודעת סיום – במקרה כזה התהליך יסיים את הריצה שלו.

כדי להבחין בין ההודעות השונות ההמתנה תתבצע באמצעות הפקודה MPI\_Probe. (חייבים את הגרסה החוסמת, כי אין לתהליך מה לעשות אם הוא לא קיבל אף הודעה)