**Minimum Spanning Tree - Prim Algorithm**

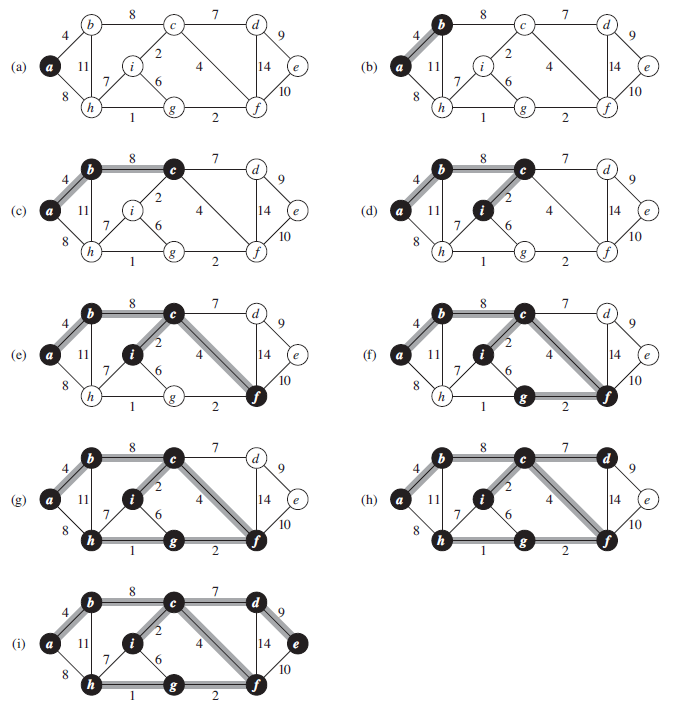
**האלגוריתם של פרים** הוא אלגוריתם **חמדני** המשמש למציאת עץ פורש מינימלי בגרף משוקלל לא מכוון.

**רעיון האלגוריתם:** יצירת עץ תוך שמירת תכונה של קשירות הגרף. האלגוריתם מתחיל את בניית העץ מקדקוד פתיחה שנבחר באקראי. בכל צעד האלגוריתם מוסיף לעץ את הצלע בעלת המשקל המינימלי מבין אלו היוצאות מקדקודי העץ ולא סוגרות מעגל.

**שלבי האלגוריתם:**

1. בוחרים באקראי קדקוד כלשהוs ומוסיפים אותו לעץ: Initialization: T = {s}
2. כל עוד T מכיל צלעות פחות מ-n-1 קודקודים:
   1. מוסיפים ל-T צלע בעלת משקל מינימאלי, כך שבדיוק קדקוד אחד נמצא ב-T וקדקוד אחר לא ב-T. כלומר,

T = T∪{a,b}: ((a∈T and b∉T) or (a∉T and b∈T)) and weight(a,b)→min

**דוגמה:**

**הרצת האלגוריתם:**

קודקוד a הוא שורש.

כל הקודקודים אשר לא נמצאים בעץ T ינהלו בתור העדיפויות המינימלית כאשר מפתח key הוא משקל מינימלי של צלע שמחברת את הקודוקוד שמחוץ לעץ לקודקוד בעץ. בהתחלה תור יכיל כל הקודקודים. בסיום הרצה תור יהיה ריק.

נחזיק מערךP של קדקודי אבות (parent vertices).

סימונים:   
משבצת אפורה – קדקוד עזב את התור Q.  
אות אדומה – מצב הקדקוד השתנה בהשוואה לשלב הקודם.

**אתחול ← שלב 1**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| קדקודים | Q | P | key |  | קדקודים | Q | P | key |
| a | a | NIL | 0 | a |  | NIL | 0 |
| b | b | NIL | ∞ | b | b | a | 4 |
| c | c | NIL | ∞ | c | c | NIL | ∞ |
| d | d | NIL | ∞ | d | d | NIL | ∞ |
| e | e | NIL | ∞ | e | e | NIL | ∞ |
| f | f | NIL | ∞ | f | f | NIL | ∞ |
| g | g | NIL | ∞ | g | g | NIL | ∞ |
| h | h | NIL | ∞ | h | h | a | 8 |
| i | i | NIL | ∞ | i | i | NIL | ∞ |

**שלב 2 ← שלב 3**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| קדקודים | Q | P | key |  | קדקודים | Q | P | key |
| a |  | NIL | 0 | a |  | NIL | 0 |
| b |  | a | 4 | b |  | a | 4 |
| c | c | b | 8 | c |  | b | 8 |
| d | d | NIL | ∞ | d | d | c | 7 |
| e | e | NIL | ∞ | e | e | NIL | ∞ |
| f | f | NIL | ∞ | f | f | c | 4 |
| g | g | NIL | ∞ | g | g | NIL | ∞ |
| h | h | a | 8 | h | h | a | 8 |
| I | i | NIL | ∞ | i | i | c | 2 |

**שלב 4 ← שלב 5**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| קדקודים | Q | P | key |  | קדקודים | Q | P | key |
| a |  | NIL | 0 | a |  | NIL | 0 |
| b |  | a | 4 | b |  | a | 4 |
| c |  | b | 8 | c |  | b | 8 |
| d | d | c | 7 | d | d | c | 7 |
| e | e | NIL | ∞ | e | e | f | 10 |
| f | f | c | 4 | f |  | c | 4 |
| g | g | i | 6 | g | g | f | 2 |
| h | h | i | 7 | h | h | i | 7 |
| I |  | c | 2 | i |  | c | 2 |

**שלב 6 ← שלב 7**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| קדקודים | Q | P | key |  | קדקודים | Q | P | key |
| a |  | NIL | 0 | a |  | NIL | 0 |
| b |  | a | 4 | b |  | a | 4 |
| c |  | b | 8 | c |  | b | 8 |
| d | d | c | 7 | d | d | c | 7 |
| e | e | f | 10 | e | e | f | 10 |
| f |  | c | 4 | f |  | c | 4 |
| g |  | f | 2 | g |  | f | 2 |
| h | h | g | 1 | h |  | g | 1 |
| i |  | c | 2 | i |  | c | 2 |

**שלב 8 ← שלב 9**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| קדקודים | Q | P | key |  | קדקודים | Q | P | key |
| a |  | NIL | 0 | a |  | NIL | 0 |
| b |  | a | 4 | b |  | a | 4 |
| c |  | b | 8 | c |  | b | 8 |
| d |  | c | 7 | d |  | c | 7 |
| e | e | d | 9 | e |  | d | 9 |
| f |  | c | 4 | f |  | c | 4 |
| g |  | f | 2 | g |  | f | 2 |
| h |  | g | 1 | h |  | g | 1 |
| i |  | c | 2 | i |  | c | 2 |

**התוצאה**:   
קבלנו עץ פורש מינימאלי שמשקלו 37 (ניתן לבדוק לפי האיור או לפי הסכום של שדות key).

לפי מערךP של קדקודי אבות (parent vertices) ניתן לבנות עץ:

**Prim - pseudo code**

**Prim(G,** root**)**

Edge T[n-1]<-empty tree //(array of edges)

numEdges = 0

**for each** v in V(G, root) // O(n)

visited[v] = false

key[v] = infinity

parent(v) = NIL

**end-for**

key[root] = 0

Q <- V(G) // Q - Min Heap, Q keyed by key[v] - O(n)

**while** (Q != empty && numEdges<n-1)

u = Q.extractMin()//O(log2(n))

**for each** v in Adj(u)

**if (**visited[v]==false && key]v[ > weight(u,v))

key]v[ = weight (u,v)

parent ]v[ = u

Q.decreaseKey(v, weight(u,v))//O(log2(n))

**end-if**

**end-for**

visited[u] = true

x=Q.getMin()

T[numEdges++]=(parent[x], x)

**end-while**

**end-Prim**

**סיבוכיות:**

בעת מימוש האלגוריתם נעשה שימוש בערימה שמתוכה מוציאים בכל פעם את הצלע המינימלית. אם משתמשים בערימה בינארית binary min-heap)) סיבוכיות האלגוריתם תהיה **O(|V|log2(|V|) + |E|\*log2|V|)=O(|E|\*log2|V|)**, כאשר ||E הוא מספר הצלעות ו-V|| הוא מספר הקדקודים.

בערימה פיבונאצ'י ניתן להגיע לסיבוכיות **O(|E| + |V|log2(|V|))**.

**השוואת האלגוריתמים לחיפוש עץ פורש מינימלי:**

כאשר גרף מכיל מספר צלעות לא גדול מומלץ להשתמש באלגוריתם מחיקת הצלעות (קרוסקל הפוך). אם בגרף יש מספר עצום של הצלעות אז לבחור באלגוריתם קרוסקל או פרים.

באופן כללי היעילות של האלגוריתם של פרים טובה מזו של האלגוריתם של קרוסקל. למרות זאת, אם הקלט כבר ממוין לפי משקלי הצלעות או כאשר ניתן למיין אותם בזמן לינארי, אזי האלגוריתם של קרוסקל יהיה מהיר יותר.

הסיבוכיות של אלגוריתם של קרוסקל ללא מיון היא O(|V|\*log2|V|). הסיבוכיות של אלגוריתם של קרוסקל עם מיון היא מורכבת מסיבוכיות של מיון O(|E|\*log2|V|) וסיבוכיות של אלגוריתם עצמו O(|V|\*log2|V|):  
O(|V|) + O(|V|log2(|V|)) + O(|E|log2(|V|)) = O(|E|log2(|V|))

**הוכחת נכונות של אלגוריתם של פרים:**

נוכיח שבכל שלב שאנו מוסיפים צלע חדשה ל-T, אנו מקבלים תת-עץ של עץ פורש מינימאלי כלשהו M.

הוכחה באינדוקציה:

א) בסיס. בשלב ראשון של האלגוריתם אנו מוציאים מתור עדיפויות Q(min heap) את   
 שורש העץ (root), שהוא שייך לכל עץ פורש מינימאלי.

ב) נניח שבשלב כלשהו של פרים קבלנו תת-עץ M של T.

ג) בשלב הבא של פרים אנו מוסיפים צלע e ל-T. אם e∈M אז גם T∪{e=(a,b)}⊆M ובזה סיימנו את ההוכחה. נניח ש-e**∉**M. נוסיף e ל-M, נקבל מעגל. במעגל זה יש מסלול שמחבר את קדקוד a עם b וכוון שרק קדקוד אחד של e שייך ל-T, (כי אלגוריתם של פרים מוסיף את e) במעגל זה קיימת צלע g=(x,y) כלשהי שרק קדקוד אחד שלה שייך ל-T. אלגוריתם של פרים היה יכול להוסיף g ל-T, אבל הוא בחר ב-,e כלומר weight(e) ≤ weight(g). נבנה עץ חדש M′=M∪{e}-{g} שמשקלו קטן או שווה למשקל של M. אבל M הוא בעל משקל מינימאלי, לכן weight(M) = weight(M′) ו- M′הוא גם עץ פורש מינימאלי שמכיל את T: T∪{e}⊆M′. מש"ל.