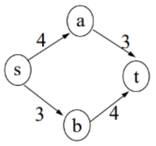
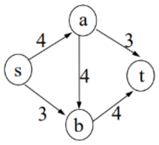
סיכום אלגוריתם מתקדם





בשני המקרים הזרימה המקסימלית היא 7

<u>זרימה ברשתות</u>

<u>רשת זרימה</u>- גרף מכוון עם קודקודים מיוחדים s, t המסמנים מקור ובור. לכל קשת (u,v) יש ערך $C(u,v) \geq 0$ שמסמן את הקיבול של הקשת.

<u>בעיית זרימה מקסימלית</u>- מהו הקצב המקסימלי בו אפשר להזרים מs לt בכפוף לקיבול על הקשתות.

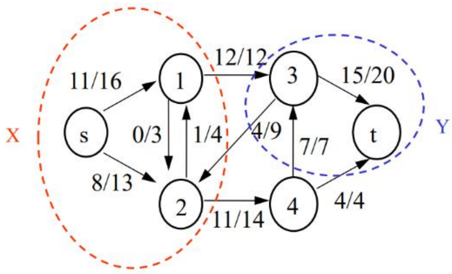
<u>פונקציית קיבול</u>- $C:V\times V\rightarrow[0,\infty)$ פונקציה המתאימה לכל זוג צמתים את הכמות המקסימלית שיכולה לזרום ביניהם.

<u>פונקציית זרימה</u>- $f: V\times V \rightarrow \mathbb{R}$ פונקציה המתאימה לכל זוג צמתים (u,v) את הזרימה המקסימלית מu לv (יש חשיבות לסדר) שמתקיים התנאים הבאים:

1. אנטי סימטריות- $f(u,v) = -f(v,u)$
2. שמירה על אילוץ הקיבול- $f(u,v)\leq C(u,v)$
3. שימור הזרימה- לכל $u\neq s,t$ מתקיים $\sum_{v\in V} f(u,v) = 0$ ← נכון משום שכל קשת מתאפסת עם החיבור לקשת ההופכית לה (אנטי סימטריות)

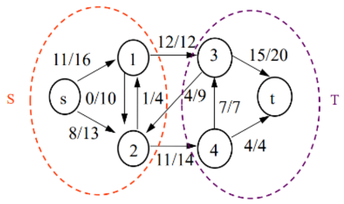
<u>קשת רוויה</u>- קשת בה $C(u,v)=f(u,v)$.

נרחיב את ההגדרה של פונקציית קיבול ופונקציית זרימה כך שנוכל להפעיל אותם על קבוצות



$C(X,Y) = C(1,3) = 12$

$f(X,Y) = f(1,3) + f(2,3) = 12 + (-4) = 8$



<u>חתך</u>- זוג קבוצות S, T יקראו חתך ברשת אם הוא מחלק את V לשתי קבוצות זרות $S,\ T=V\setminus S$

<u>קיבול חתך</u>- סכום הקיבולים החוצים את החתך $C(S,T) = \sum_{u\in S}\sum_{v\in T} C(u,v)$

<u>זרימה בחתך</u>- סכום הזרימות החוצות את החתך $f(S,T) = \sum_{e\in(S,T)} f(e) - \sum_{e\in(T,S)} f(e)$

$f(S,T)\leq C(S,T)$

<u>למת החתך</u>- כאשר f זרימה כלשהי ו(S,T) חתך כלשהו $|f|=f(S,T)$ כלומר סך הזרימה בחתך שווה לסך הזרימה בגרף.

<u>קשת משפרת</u>- קשת (u,v) שאפשר להזרים דרכה יותר מאשר מוזרם בה כרגע (לא רוויה או שמוזרם משהו בכיוון הנגדי).

<u>מסלול משפר</u>(1)- מסלול מs לt שמורכב כולו מקשתות משפרות.

<u>קיבול שיורי</u> - קיבול שיורי C_f בין זוג קודקודים הוא המידה בה ניתן להגדיל את כמות הזרימה ביניהם.

<u>גרף שיורי</u>- גרף שיורי G_f מכיל את כל הקשתות שקיבולן השיורי חיובי.

<u>מסלול משפר</u>(2)- מסלול בגרף השיורי מs לt.

<u>קיבול של מסלול</u>- הקיבול המינימלי לאורך המסלול.

<u>קיבול שיורי של מסלול</u>- הקיבול המינימלי לאורך המסלול ברשת השיורית.

P מסלול משפר אם ורק אם הקיבול השיורי של P חיובי.

<u>Max Flow, Min Cut</u>-

f זרימה ברשת G, התנאים הבאים שקולים:

1. f זרימה מקסימלית בG.
2. בגרף G_f אין מסלולים מs לt
3. קיים חתך (S,T) שעבורו $|f|=C(S,T)$ (חתך רווי).

שיטת פורד-פלקנסון

השיטה הבסיסית:

קלט:

רשת זרימה G פונקציית קיבול C, מקור s ובור t.

אתחול:

זרימה 0 בכל הקשתות- $f(u,v)=f(v,u)=0$

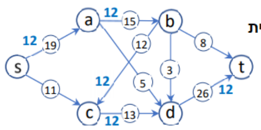
הגרף השיורי G_f שווה לגרף המקורי G- $C_f(u,v) = C(u,v)$

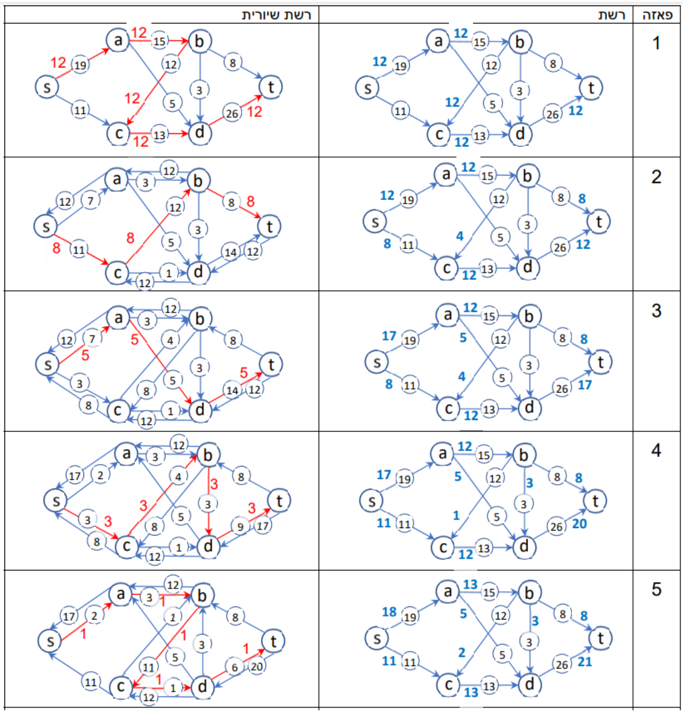
האלגוריתם:

1. כל עוד יש מסלול משפר (מסלול מs לt בגרף G_f):
   1.1. נמצא מסלול כזה P ונחשב את קיבול השיורי $C_f(P)$
   1.2. נגדיל את הזרימה בG לאורך P ב$C_f(P)$ כך:
      1.2.1. לכל קשת (u,v) במסלול P $f(u,v) = f(u,v) + C_f(P)$
   1.3. נעדכן את G_f לאורך P כך:
      1.3.1. לכל קשת (u,v) במסלול P נעדכן את הקיבול השיורי של הקשת.

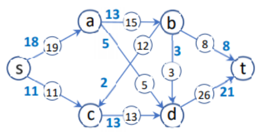
דוגמה עם בחירת מסלול בשיטה חמדנית:

רשת מקורית





רשת סופית



אפשר לראות שברשת השיורית לא קיימים עוד מסלולים מS לT משמע אין מסלול משפר

ברשת בה הקיבולים שלמים האלגוריתם בטוח יעצור ויחזיר את הזרימה המקסימלית. בגרף שבו הקיבולים ממשים כלשהו לא בטוח שהאלגוריתם יעצור אבל נוכל להשתמש באדמונד קארפ או דיניץ

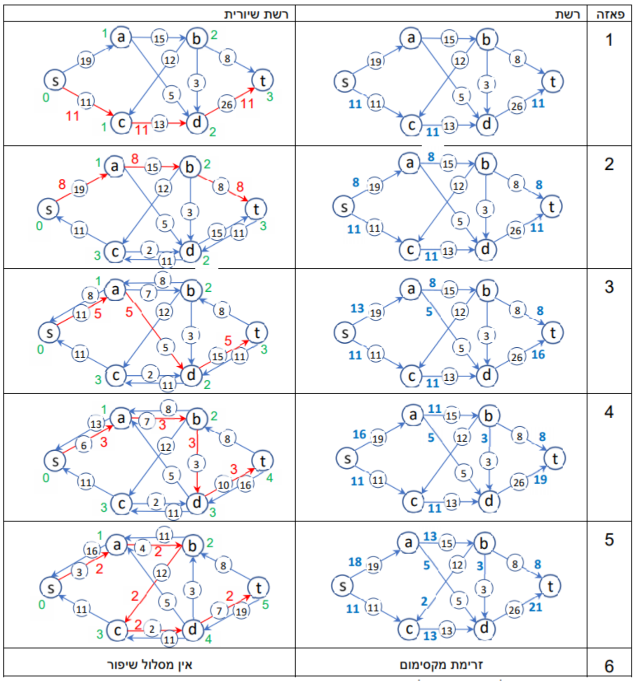
סיבוכיות: בכל איטרציה בונים גרף שיורי וממנו מוציאים מסלול לשיפור הגרף המקורי ז"א O(E) השאלה כמה איטרציות נעשו.

דוגמאות לשיטות בעמוד הבא

בשיטת EK (אדמונד קארפ)- נשתמש בBFS כדי למצוא את המסלול שמספר הקשתות שלו הכי קטן- מספר איטרציות V*E ולכן הסיבוכיות $O(E^2V)$.

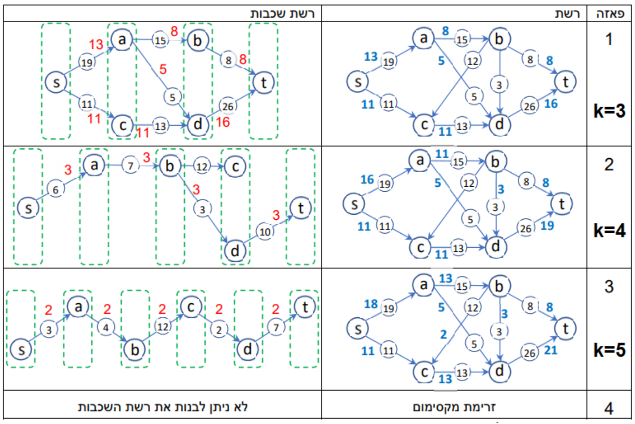
בשיטת דיניץ- נגדיר משתנה k=1 שיגדל ב1 בכל איטרציה ובכל איטרציה נמצא את כל המסלולים בעלי אורך K שאינם מתנגשים ונעדכן את כולם יחד בגרף המקורי – מספר איטרציות V^2 ולכן הסיבוכיות $O(V^2E)$.

אדמונד קארפ



דיניץ

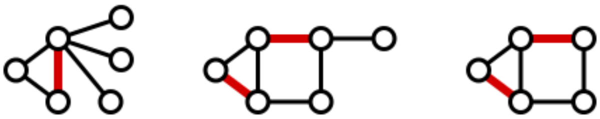
בכל פאזה, לא שורטטה רשת שיורית אלא רק רשת שכבות.



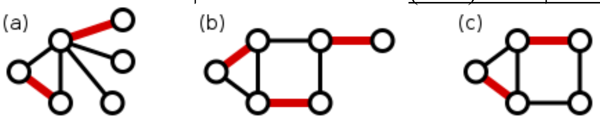
<u>זיווגים</u>

<u>זיווג</u>- אוסף של קשתות מאותו הגרף כך שאין שתי קשתות באוסף שנוגעות בצומת משותף.

<u>זיווג מקסימלי</u>- שידוך שלא ניתן להגדיל אותו ע"י הוספת קשתות נוספות לזיווג



<u>זיווג מקסימום (מרבי)</u>- זיווג M בעל מס' הקשתות הרב ביותר בגרף ולכל שידוך אחר M' מתקיים $|M|\geq|M'|$.



***כל זיווג מקסימום הוא גם זיווג מקסימלי אבל לא להפך**

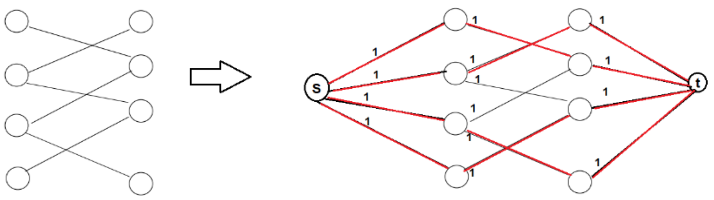
<u>זיווג מושלם</u>- זיווג שבו משתתפים כל הצמתים בגרף, כלומר חלוקה של הצמתים לזוגות (כמו גרפים דו צדדים).

ניתן לבדוק שזיווג הוא מושלם ע"י יצירת גרף דו צדדי בו כל צד מכיל את כל הקודקודים והוספת קודקודי s ו-t.

s יקושר לכל הקודקודים בצד ימין, t יקושר לכל הקודקודים בצד שמאל והקודקודים בימין ושמאל יהיו מקושרים ביניהם לפי הקשתות בגרף המקורי (אם הם מקושרים בגרף הם יהיו מקושרים גם פה).

נגדיר את פונקציית הקיבול של כל קשת ל-1.

נחפש זרימה מקסימלית בגרף ואם הזרימה שווה למס' הקודקודים בגרף המקורי- יש לנו זיווג מושלם.



<u>תכנות דינמי</u>

צמיחה מלמטה למעלה כך שאנו פותרים את אותה בעיה ע"י הסתמכות על זיכרון של תוצאות קודמות.

דוגמה לפתרון בעיית התרמיל:

	נפח (w_i)	ערך (b_i)
קואלה	6	30
דוב	3	14
ארנבת	4	18
חתול מוזר	2	9

	0	1	2	3	4	5	6	7	8	9	10
$w_1=6$ $b_1=30$	0	0	0	0	0	0	30	30	30	30	30
$w_2=3$ $b_2=14$	0	0	0	14	14	14	30	30	30	44	44
$w_3=4$ $b_3=18$	0	0	0	14	18	18	30	32	32	44	48
$w_4=2$ $b_4=9$	0	0	9	14	18	23	30	32	39	44	48

העמודות מייצגות את ההגבלה שלנו (במקרה זה מה המשקל שאנחנו יכולים לקחת).

השורות מייצגות את האובייקטים שאנחנו יכולים להשתמש בהם כאשר בסוף כל שורה נוסיף מוצר חדש ונראה האם הוא נותן לנו תוצאה טובה יותר.

התאים מייצגים את הערך המקסימלי הנוכחי שהגענו אליו.

ובטבלה נוספת בגודל זהה (או ע"י הוספת משתנה לתא של הטבלה) נשמור מידע לגבי הם האיבר בשורה ה i משפר את התוצאה שהתקבלה ע"י כל האיברים הקודמים לו כאשר ההגבלה שלנו היא העמודה j.

לבסוף נשחזר את התוצאה ע"י בדיקה איזה איברים גרמו לשיפור

	0	1	2	3	4	5	6	7	8	9	10
$w_1=6$ $b_1=30$	0	0	0	0	0	0	30	30	30	30	30
$w_2=3$ $b_2=14$	0	0	0	14	14	14	30	30	30	44	44
$w_3=4$ $b_3=18$	0	0	0	14	18	18	30	32	32	44	48
$w_4=2$ $b_4=9$	0	0	9	14	18	23	30	32	39	44	48

לא היה שיפור

לא היה שיפור

j

6=10-4

w3

והפריטים שנלקחו תחת מגבלה j=10 הםi=0,i=2

וכצפוי ערכם המשולב הוא 48

<u>התאמת מחרוזות</u>

<u>סימנים</u>

$\sum$ - אלף-בית קבוצה סופית של תווים.

$x \cdot w$– שרשור המילה x אחרי המילה w

a^n - שרשור של המילה a על עצמה n פעמים

$|w|$- האורך של המילה w

$\sum^i$- קבוצת כל המילים באורך i שבנויות מהאלף-בית

Q – קבוצה סופית של מצבים

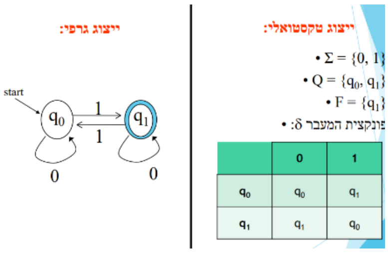
q_0- מצב תחילי מתוך Q

F- קבוצת מצבים סופיים

δ- פונקציות מעברים $\delta: Q\times\Sigma \to Q$ (המצב הבא=(תו, מצב נוכחי)δ)

$\hat{\delta}$- הגדרת עזר שנוכל להיעזר בה כדי לבטא את המצב אליו האוטומט יגיע ממצב כלשהו עם מילה כלשהי (אותו דבר כמו פונקציית מעברים רק שפה התייחסנו למילה ולא לתו יחיד מטעמי נוחות).

<u>מושגים</u>



	0	1
q_0	q_0	q_1
q_1	q_1	q_0

אוטומט- מודל לסוג מסוים של אלגוריתמים על מחרוזות שנוח לייצג כדיאגרמת מצבים.

אוטומט סופי דטרמיניסטי- מאופיין ע"י $Q, q_0, F, \Sigma, \delta$

<u>חישוב של אוטומט</u>

האוטומט בהתחלה נמצא בתחילת סרט הקלט במצב תחילי (q_0) וע"י קריאה מסרט הקלט הוא משנה את מצבו לפי פונקציות המעברים (δ) ומתקדם לתו הבא.

נגיד שאוטומט מקבל\מזהה מחרוזת w אם $\hat{\delta}(q_0, w) \in F$.

קבוצת המילים שהאוטומט מזהה נקראת השפה

<u>הוכחה שאוטומט מזהה שפה</u>

1. מגדירים את תפקידי המצבים- לכל מצב נאפיין את קבוצת המילים המגיעה אליו.
2. מוכיחים שמילה מגיעה למצב הנכון ע"י אינדוקציה.
3. מוכיחים את נכונות המצבים המזהים- נראה שכל המילים השייכות לשפה L הן בדיוק המילים שמגיעות לF.

<u>בעיית התאמת מחרוזת ע"י אוטומט</u>

בבעיה זו נרצה לבנות אוטומט כך שנגדיר את כל המצבים והמעברים האפשריים במכונה- נתייחס למצב כאל מיקום האות הנבדקת וקביעת המעברים יתבצע בהתאם למחרוזת הרצויה.

```
COMPUTE-δ(String p[1...m])
δ[0,P[1]] ← 1
for each c∈Σ, c ≠ P[1] do
    δ[0,c] ← 0
state ← 0
for i ← 1,...,m−1 do
    δ[i,P[i+1]] ← i+1
    for each c∈Σ, c ≠ P[i+1] do
        δ[i,c] ← δ[state, c]
    state ← δ[state, P[i+1]]
for each c∈Σ do
    δ[m,c] ← δ[state, c]
```

הגדרת מעבר ממצב 0 ל1 עבור קלט זהה לזה שבP[1] (בעצם פה מתחילה הבדיקה)

הגדרת מעבר ממצב 0 למצב 0 עבור כל קלט שונה מP[1]

באותו אופן נגדיר את המעברים עבור מצב i

הגדרת המעבר מהמצב הm למצב מקבל עבור כל קלט (במצב זה "הצלחנו" לעבור על כל המחרוזת)

סיבוכיות- $\sum_{i=1}^{m} |\Sigma| = \Theta(|\Sigma| \cdot m)$

אלגוריתם KMP הדרך של דוד

באלגוריתם זה נבצע עבודה חלקית בשלב הבניה של האוטומט ונשלים את החישוב תוך כדי השימוש בו.

<u>בזמן הבניה</u>

נשמור בטבלה state[1,…,m] את ערכי המשתנה state מהגרסה הקודמת כך שstate[i] = ערכו של המשתנה state בתחילת האיטרציה הi = $\hat{\delta}(q_0, P[2..i])$.

<u>כשצריך את δ</u>

נקרא לפונקציה שמחשבת את המעבר הנוכחי δ(i,c) במקום לשלוף את הערך מטבלה.

```
Search(char T[1..n], Transition Function δ)
s ← 0        // initial state
for i ← 1,...,n do
  s ← δ(s, T[i])
  if s = m then
    print Match in position i – m + 1
```

```
δ(State i, Character c)
if c = p[i+1] then
   return i + 1
else if i = 0 then
   return 0
else
   return δ(state[i],c)
```

<u>שלב בניית האוטומט</u>

מספר הקריאות לפונקציה δ הוא m_כאורך התבנית

```
SetState(String p[1...m])    // Fill state array.
state[1] ← 0
for i ← 1,...,m-1 do
   state[i+1] ← δ(state[i], p[i+1])
```

<u>תיאור האלגוריתם</u>

1. נבנה את המערך state ע"י הפונקציה SetState.
2. נריץ את האוטומט על הטקסט T כקלט בעזרת הפונקציה Search ופונקציית המעברים δ שכתבנו.

יעילות כוללת של האלגוריתם- Θ(n+m).

<u>דוגמאות</u>

P=ABCDABD

setState

i	1	2	3	4	5	6	7
state[i]	0	δ(0,B)=0	δ(0,C)=0	δ(0,D)=0	δ(0,A)=1	δ(1,B)=2	δ(2,D)=0

T=ABACABABA

i		1	2	3	4
T		A	B	A	C
s	0	δ(0,A)=1	δ(1,B)=2	δ(2,A)=δ(state[2],A)=δ(0,A)=1	δ(1,C)=)=δ(state[1],C)=δ(0,C)=0

i		5	6	7	8	9
T		A	B	A	B	A
s	0	δ(0,A)=1	δ(1,B)=2	δ(2,A)=δ(state[2],A)=δ(0,A)=1	δ(1,B)=2	δ(2,A)=δ(state[2],A)=δ(0,A)=1

לא הגענו לs=7 ולכן לא קיימת תת מחרוזת P בT

P=BABBABCAB

setState

i	1	2	3	4	5	6	7	8	9
state[i]	0	δ(0,A)=0	δ(0,B)=1	δ(1,B)= δ(0,B)=1	δ(1,A)=2	δ(2,B)=3	δ(3,C)= δ(state[3],C)= δ(1,C)=0	δ(0,A)=0	δ(0,B)=1

T=ABABABABBABCAA

i	1	2	3	4	5	6	7
T	A	B	A	B	A	B	A
s	δ(0,A)=0	δ(0,B)=1	δ(1,A)=2	δ(2,B)=3	δ(3,A)= δ(state[3],A)= δ(1,A)=2	δ(2,B)=3	δ(3,A)= δ(state[3],A)= δ(1,A)=2

i		8	9	10	11	12	13	14
T		B	B	A	B	C	A	A
s	2	δ(2,B)=3	δ(3,B)=4	δ(4,A)=5	δ(5,B)=6	δ(6,C)=7	δ(7,A)=8	δ(8,A)= δ(state[8],A)= δ(0,A)=0

לא הגענו לs=9 ולכן לא קיימת תת מחרוזת P בT

סיבוכיות

הגדרות

אלגוריתם הכרעה- אלגוריתם A מכריע שפה L = $A(x)=\begin{cases}1 & x\in L\\ 0 & x\notin L\end{cases}$ R היא המחלקה של כל אלגוריתמי ההכרעה

אלגוריתם אימות- אלגוריתם A מאמת שפה L = לכל x∈L קיים אישור y כך שA(x,y)=1 ולכל x∉L לא קיים אישור y כך שA(x,y)=1

מחלקות סיבוכיות

המחלקה P:

המחלקה L מוכרעת בזמן פולינומיאלי על ידי אלגוריתם A – קיים קבוע k כך שA מכריע כל קלט x בזמן $O(|x^k|)$.

דוגמאות לשפות בP- מציאת מסלול בגרף, מסלולים הקצרים, BFS, DFS, מיון, שידוך, זרימה...

תכונות במחלקה-

המחלקה NP:

המחלקה L מאומתת בזמן פולינומיאלי על ידי אלגוריתם A – קיים קבועים c, k כך לכל קלט x∈L קיים אישור y, $|y| = O(|x^k|)$ ואלגוריתם A מאמת בזמן $O(|x^k|)$ כי A(x,y)=1 ולכל x∉L A(x,y)=0.

הסבר- בעיות שלא הצלחנו לפתור בזמן פולינומיאלי אבל בהינתן מצב ספציפי (שמאמת אותן) ניתן לפתור אותה בזמן פולינומיאלי

למשל

1)לא נוכל לקבוע האם גרף הוא גרף המילטוני אבל בהינתן מסלול (בתקווה מעגל המילטון) ניתן לבדוק האם הגרף המילטוני.

2)SAT(האם CNF ספיקה)- לא מצאנו אלגוריתם פולינומיאלי אבל בהינתן השמה/ניחוש (בתקווה השמה שמקיימת את התנאי) ניתן לבדוק אם היא מספקת את הנוסחה.

המחלקה CO-NP:

עולם בעיות שגם אימות לא מאפשר פתרון שלהם אבל ניתן לבדוק שלילה שלהם בזמן פולינומיאלי בהינתן מצב ספציפי (שמפריך אותם) כמו טאוטולוגיה כשיש צורך לבדוק את כל ההשמות או אם ננחש השמה אחת שבה נקבל false.

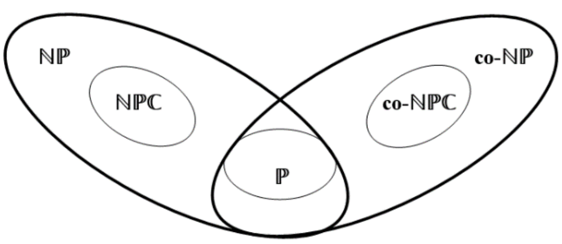
שלמה-NPC/NP:

שפה L נקראת NPC אם:

1. L∈NP
2. עבור כל L'∈NP מתקיים $L' \le_P L$

*שפה המקיימת את תכונה 2 בלבד היא NP-קשה

מאמינים שהיחס בין הקבוצות $\mathbb{P}$, $\mathbb{NP}$, co-$\mathbb{NP}$, $\mathbb{NPC}$, co-$\mathbb{NPC}$



<u>רדוקציה</u>

יהיו P1 וP2 בעיות, נאמר שבעיה P1 ניתנת לרדוקציה לבעיה P2 אם קיים אלגוריתם f כך שx הוא הוא פתרון של P1 אםם f(x) הוא פתרון של P2.

אם f היא פונקציה חשיבה בזמן פולינומיאלי וגם כל x של P1 מתקיים f(x) הוא פתרון של P2 נגיד ש

$P1 \leq_P P2$ ← P1 יותר קלה מP2 ופתרון של P2 גורר פתרון של P1 (אבל לא להפך).

דוגמה פשוטה:

ברצוננו למצוא מספר מינימלי במערך ובידינו אלגוריתם למציאת מס' מקסימלי במערך.

הרדוקציה היא מהבעיה שאנו רוצים לפתור (P1) (מציאת מינימום) לבעיה שאנחנו יודעים לפתור (P2) (מציאת מקסימום).

פונקציית הרדוקציה במקרה זה תהיה הכפלת הערכים במערך ב1-, מציאת הפתרון לבעיית מקסימום(P2) והחזרת הפלט כפול 1-.

הוכחת נכונות:

עלינו להראות שהרדוקציה לא הוסיפה פתרון ולא איבדה פתרון.

כיוון 1 לא אבד פתרון נכון- תשובה נכונה נשארת נכונה

כיוון 2 לא נוסף פתרון- תשובה של הרדוקציה מתורגמת לתשובה נכונה של הבעיה.

כאשר נרצה להראות שבעיה נמצאת בNPC לאחר הוכחה שהיא אכן בNP נרצה להראות שהיא יותר קשה מבעיה אחרת בNPC ← ז"א נעשה רדוקציה מבעיה בNPC לבעיה שלנו.

דוגמה בNPC:

IS (independnt set) - קבוצת קודקודים בלתי תלויים בגרף –אין קשתות בין שום קודקוד לאחר שבקבוצה.

נוכיח IS בNPC עם רדוקציה מISל VC:

1. בהינתן עד (קבוצת קודקודים S) בגודל K נבדוק שאין קשת בין אף אחד מהקודקודים בקבוצה (זמן פולינומי).

2. נראה רדוקציה מ- IS ל-VC (נפתור את Vertex Cover באמצעות Independent Set):

בהינתן גרף G(V,E) וגודל K נגדיר את פונקציית הרדוקציה F(G(V,E),k) → (G(V,E),v-k).

אם קיים VC בגודל K בגרף G – אז כל הקודקודים האחרים V-K בגרף מהווים VC, מכיוון שמכל קודקוד ב-IS יש לפחות קשת אחת לקודקוד שאינו ב-IS (אחרת היו מנותקים מהגרף),

ולכן, אם נבחר את כל הקודקודים שלא ב-IS אז כיסינו את כל הקשתות.

מכאן שניתן למצוא VC בגודל V-K.

F מלאה – לכל גרף ולכל K ניתן לחשב VC.

F חשיבה – ניתן לבנות מכונה שמבצעת את הרדוקציה בזמן פולינומי.

F פרידה – אם (G(V,E),k) שייכת ל-IS אז (G(V,E),v-k) שייכת ל-VC. ואם לא אז לא (לא אבד פיתרון ולא נוסף).

<u>סיבוכיות מקום</u>

ניתוח המשאבים שאלגוריתם דורש- במטא את כמות הזיכרון שהאלגוריתם נחוץ לו בזמן הריצה
אפשר להגדיר חישוב לא דטרמיניסטי גם בעולם המקום- האלגוריתם מקבל ניחוש כחלק מהקלט וצריך לבדוק אותו.

PS- מחלקת הבעיות הבוליאניות מוכרעות ע"י מכונת טיורינג דטרמיניסטית המשתמשת בנפח פולינומיאלי.

NPS- מחלקת הבעיות הבוליאניות מאומתות ע"י מכונת טיורינג לא דטרמיניסטית המשתמשת בנפח פולינומיאלי.

NPSPACE=PSPACE

CO-PSPACE=PSPACE בגלל שנריץ במקום פולינומי ונחזיר את התשובה ההפוכה.

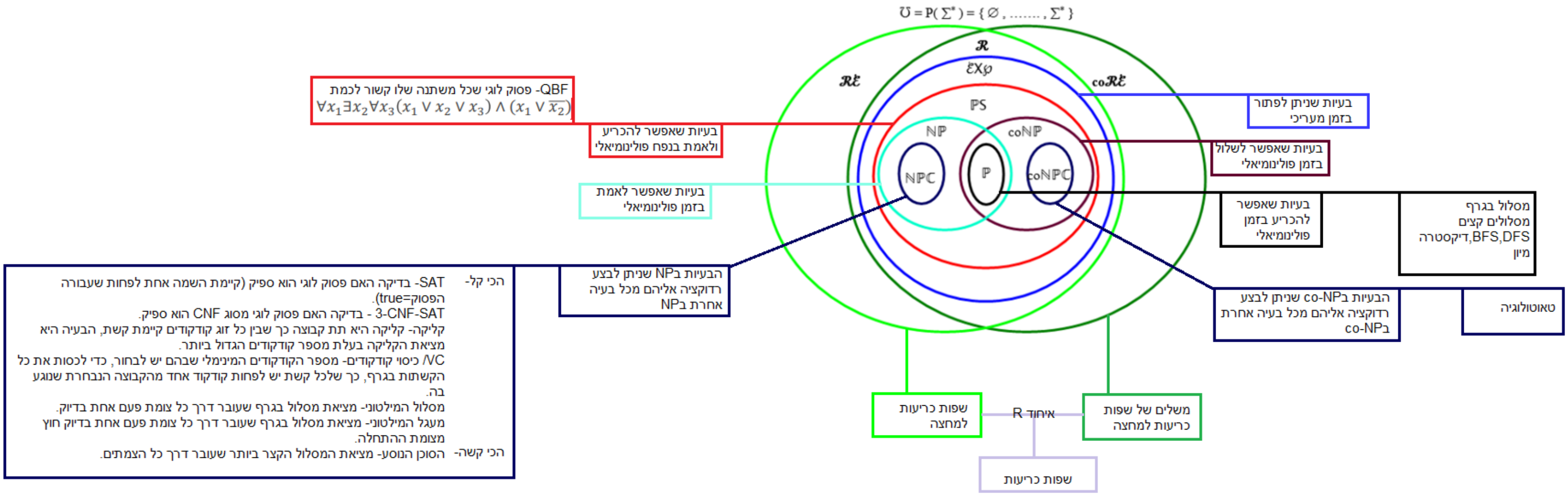
בבעיות אלו סיבוכיות הזמן אינה מוגבלת.

דוגמה:

בעיית QBF היא בעיה בה נתון פסוק לוגי שלכל משתנה שלו יש כמת למשל.

$$\forall x_1 \exists x_2 \forall x_3 (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2})$$

בבעיה זו עלינו לבדוק את כל המצבים וגם לשמור את הערכים במסלול הנוכחי שאנו בודקים (נוכל לזרוק אותו בסוף המעבר ולשמור במקום מסלול חדש שנבדק) n משתנים ← נפח פולינומי בזיכרון וזמן ריצה אקספוננציאלי.



<u>אלגוריתם קירוב</u>

אלגוריתם קירוב הוא פתרון שאינו בהכרח אופטימלי לבעיה אבל קרוב אליה ושימושי בעיקר עבור בעיות בNP משום שהסיבוכיות שלהם גבוה.

אלגוריתם A הוא בעל יחס קירוב p (p≥1) אם עבור כל קלט היחס בין העלות C של הפתרון שמפיק A לעלות הפתרון האופטימלי *C מקיים:

לבעיית מקסימום: $\frac{C*}{C} \leq p$ לבעיית מינימום: $\frac{C}{C*} \leq p$

```
APROX-VERTEX_COVER(G)
C ← ∅
E' ← E
while E' ≠ ∅
    do let (u,v) be an arbitary edge of E'
        C ← C∪{u,v}
        remove from E' every edge incident on either u or v
return C
```

דוגמה:

נסמן בA את קבוצת הקשתות שנבחרה בשורה 4.

קבוצה זו אינה מכילה קשתות שלהן צמתים משותפים ← |C|=2|A|.

הפתרון האופטימלי *C מכסה את הקשתות A ולכן חייב להכיל לפחות צומת אחד לכל קשת ← |C*|≥|A|

ולכן |C| ≥|2|C* , VC זו בעיית מינימום (מחפשים מספר קודקודים מינימלי) - $p \geq 2$ $\quad p \geq \frac{C}{C*} \leq \frac{2A}{A} = 2$

קיבלנו יחס קירובי 2.

<u>אלגוריתם מקוון</u>

אלגוריתם שלא מקבל את כל הקלט שלו מראש, אלה תוך כדי ריצה ועליו לקבל החלטות במצב של חוסר ודאות.
המדד להצלחת האלגוריתם הוא יחס התחרותיות והוא נעשה בהשוואה לאלגוריתם לא מקוון (offline).
למשל- מערכת ההפעלה בוחרת אילו דפים לקחת מהזיכרון האיטי לזיכרון המהיר מבלי לדעת אילו דפים ידרשו בעתיד.

<u>יחס התחרות החזקה</u>

בהינתן אלגוריתם מקוון ON יחס התחרות החזקה הוא α אם מתקיים $ON(\sigma)\leq\alpha\cdot OPT(\sigma)$

<u>דוגמה</u>

בעיית הסקי:

עלות השכרת ציוד סקי= 100 דולר ליום.

עלות רכישת ציוד סקי= B מאות דולרים.

אננו יודעים כמה זמן תימשך חופשת הסקי- האם כדאי לקנות ציוד או להשכיר אותו?

n- מספר ימי הסקי (לא ידוע מראש)

σ- סדרת אירועים

פתרון אופטימלי (offline)- $OPT(\sigma)=\min(B,n)$

פתרון מקוון

שכור את הציוד עד היום הT וביום הT קנה אותו $ON_T(\sigma)\begin{cases} n & T>n \\ T+B-1 & T\leq n \end{cases}$

נבדוק את יחס התחרותיות החזקה של האלגוריתם:

עבור n < B - $\frac{ON_B(\sigma)}{OPT(\sigma)}=\frac{n}{n}=1$

עבור n ≥ B - $\frac{ON_B(\sigma)}{OPT(\sigma)}=\frac{B+B-1}{B}=2-\frac{1}{B}$

נטען כי $2-\frac{1}{B}$ הוא החסם התחתון ליחס התחרותיות החזקה של הבעיה ונוכיח זאת בכך שנראה כי לכל אלגוריתם מקוון ON_T מתקיים $\frac{ON_B(\sigma)}{OPT(\sigma)}\geq 2-\frac{1}{B}$ נקבע לכל T -n=T ונקבל:

$$\frac{ON_B(\sigma)}{OPT(\sigma)}=\frac{T+B-1}{\min(T,B)}=\frac{T}{\min(T,B)}+\frac{B-1}{\min(T,B)}\geq\frac{T}{T}+\frac{B-1}{B}=2-\frac{1}{B}$$