האקדמית תל-אביב-יפו

מספר זהות:

תשע"ז סמסטר א', מועד ב'
תאריך: יום ד' 27.2.2017
שעה: 9:00
משך הבחינה: שלוש שעות וחצי
חומר עזר: דפי עזר מצורפים

בחינה בקורס: אלגוריתמים

מרצה: ד"ר איריס רוזנבלום

מדבקת
ברקוד

הנחיות:

- יש לענות על כל השאלות.
- כתבו את תשובותיכם על גבי טופס המבחן במקום המוקצה לכך. לא תיבדקנה תשובות שיירשמו במחברות הטיוטה.
- הקפידו על כתב מסודר וברור!
- תשובות ללא הוכחה והסבר לא תקבלנה ניקוד מלא!
- אם אתם משתמשים באלגוריתם זהה לחלוטין למה שנלמד בכיתה אפשר להשתמש בו כבקופסה שחורה. אולם אם אתם משנים משהו – תארו את השינויים במדויק.

טופס הבחינה כולל 14 עמודים

בהצלחה!

מס' ת.ז. ________________   מס' מחברת ________________.

שאלה מס' 1 (20 נק')

להלן טבלה המתארת אוטומט לזיהוי תבנית ולידו המערך state המשמש כעזר לאלגוריתם.

עליכם למלא את המקומות החסרים, כלומר הטורים 4 ו 5- . שימו לב שהתבנית אינה נתונה – עליכם לשחזר אותה ולכתוב אותה במקום המיועד לכך. יש רק פתרון אחד נכון.

מצב	0	1	2	3	4	5
A	0	2	0	2		
B	1	1	3	4		
C	0	0	0	0		

מערך state (KMP)

0	0	1	1	2

המחרוזת (השלימו):

P =

מס' ת.ז. ______________　　　　　　**מס' מחברת ______________.**

<u>מקום לחישובים:</u>

מס' ת.ז. ________________ מס' מחברת ________________.

שאלה מס' 2 (20 נק')

להלן האלגוריתם הרקורסיבי לפתרון בעיית תרמיל הגב:

```
Knapsack({s1, s2, …, sn}, W)
    if (n == 1)
        if (w1 > W)
            return 0;
        else
            return b1;
    else
        if (wn > W)
            return Knapsack({s1, s2, …, sn-1}, W);
        else
            with_last = bn + Knapsack({s1, s2, …, sn-1}, W-wn);
            without_last = Knapsack({s1, s2, …, sn-1}, W);
            return max(with_last, without_last);
```

עליכם לכתוב את הפסאודו קוד של האלגוריתם ממומש **בתכנות דינאמי**.

מס' ת.ז. ______________ מס' מחברת ______________.

אלגוריתם:

מס' ת.ז. ____________ מס' מחברת ____________.

<u>שאלה מס' 3</u> (20 נק')

נתון גרף $G=(V,E)$ לא מכוון קשיר עם משקלים על קשתותיו, וכן עץ פורש מינימלי שלו, T.

עתה מוסיפים קשת חדשה לגרף, e, עם משקל $w(e)$.

תארו אלגוריתם יעיל ככל שתוכלו למציאת עץ פורש מינימלי בגרף החדש (שמכיל את הקשת הנוספת).

הוכיחו נכונותו וחשבו סיבוכיותו.

אלגוריתם:

הוכחת נכונות:

סיבוכיות:

<u>שאלה מס' 4</u> (40 נק')

הוכיחו או הפריכו כל אחת מהטענות הבאות:

א. (13 נק') יהי G גרף מכוון, ויהי C מעגל מכוון פשוט ב- G.

טענה: כל קדקודי המעגל יופיעו בהכרח על מסלול מכוון אחד בעץ ה- DFS המתקבל.

הקיפו: **הטענה נכונה** / **הטענה לא נכונה**

הוכחה:

ב. (13 נק') מריצים את אלגוריתם בלמן פורד על גרף מכוון $G(V,E)$ עם משקלים, ועוצרים את האלגוריתם אחרי מספר כלשהו של פעולות Relax (בלי להגיע בהכרח עדיין לתנאי הסיום של האלגוריתם).

טענה: אם $p[v]=u$ אז $d[v]\geq d[u]+w(u,v)$

הקיפו: **הטענה נכונה** / **הטענה לא נכונה**

הוכחה:

ג. (14 נק') תהי f זרימה ברשת G, ותהי ' f זרימה בגרף השיורי G_f.

טענה: $f+f'$ היא זרימה ב- G

הקיפו: **הטענה נכונה** / **הטענה לא נכונה**

הוכחה:

דפי עזר

TOPOLOGICAL-SORT(Graph G)

```
Queue Q;   // Queue of sources.
int indegree[n]   // Array of indegrees of vertices

// INIT: compute in-degrees of v in indegree[v].
for each v∈ V do
        indegree[v] ← 0
for each (u,v)∈ E do
        indegree[v] ← indegree[v]+1
for each v∈ V do
        if indegree[v]=0 then
                Q.Enqueue(v)
// MAIN LOOP
while Q ≠∅ do
        v ← Q.Dequeue()
        print v
        for each u∈ Adj[v] do
                indegree[u] ← indegree[u]-1
                if indegree[u]=0 then
                        Q.Enqueue(v)
// check if all nodes reached
for each v∈V do
        if indegree[v] ≠ 0 then
                print ``No Topological Sort!"; stop
```

FIND-CIRCUIT(Vertex v_0)

```
// Find a circuit starting at v0. Return list of vertices.
v ← v0
List  L ← ⟨v0⟩                // initialize list
repeat
        u ← a neighbour of v via an unused edge
        mark (v,u) used
        L.Append(u)
        v ← u
until  v  has no unused edge
return L
```

EULER(Graph G)

```
// Find an Euler circuit.
L ← FIND-CIRCUIT(v1)

while there is a vertex in L with unused edges
        v ← first such vertex in L
        L1  ← FIND-CIRCUIT(v)
        "paste" L1  into L instead of v
return L
```

BFS(Graph G, Vertex s)

```
Queue Q;      // Queue of vertices visited.
// INIT
for each vertex v do
        d[v] ← ∞
Q ← {s}
d[s] ← 0

// MAIN LOOP
while Q ≠ ∅ do
        u ← Q.Dequeue()
        for each v ∈Adj[u]  do
                if d[v] = ∞ then
                        d[v] ← d[u] + 1
                        Q.Enqueue(v)
```

DFS(Graph G)

```
// INIT
for each vertex u do
        Color[u] ← white
// MAIN LOOP
for each vertex u do
        if Color[u] = white then
                VISIT(u)
```

VISIT(Vertex u)

```
Color[u] ← gray              // begin processing of u
for each v ∈Adj[u] do
        if Color[v] = white then
                mark edge (u,v)
                VISIT(v)
Color[u] ← black             // end processing of u
```

DIJKSTRA(Graph G, Weight w, Vertex s)

```
PriorityQueue Q

d[s] ← 0                  // INIT
p[s] ← NULL
for each vertex v ≠ s do
        d[v] ← ∞
        p[v] ← NULL

// Build priority queue
Q.Build(V, d)

// MAIN LOOP
while Q ≠ ∅ do
        u ← Q.Delete-Min()
        for each v ∈ Adj[u]
                if d[v] > d[u] + w(u,v) then
                        d[v] ← d[u] + w(u,v)
                        p[v] ← u
                        Q.Decrease-Key(v, d[v])
```

BELLMAN-FORD(Graph G, Weight w, Vertex s)

```
d[s] ← 0                  // INIT
p[s] ← NULL
for each vertex v ≠ s do
        d[v] ← ∞
        p[v] ← NULL

for i ← 1,..., n–1 do          // Main loop
        for each (u,v)∈E do
                RELAX(u,v)
for each (u,v)∈E do     // Check Termination
        if d[v] > d[u] + w(u,v) then
                return "NEGATIVE-CYCLE"
return "SUCCESS"
```

RELAX(Vertex u, Vertex v)

```
if d[v] > d[u] + w(u,v) then
        d[v] ← d[u] + w(u,v)
        p[v] ← u
```

מק"בים בגרף אציקלי

```
List L ← TOPOLOGICAL_SORT(G)
d[s] ← 0                  // INIT
p[s] ← NULL
for each vertex v ≠ s do
        d[v] ← ∞
        p[v] ← NULL

// MAIN LOOP
for each vertex u∈L do
        for each v∈Adj[u] do
                RELAX(u,v)
```

FLOYD WARSHALL(Graph G, Weight w)

```
Weight d[1..n, 1..n] ← {∞, …, ∞}
Int p[1..n, 1..n] ← {NULL,…,NULL}

// INIT
for i =1,…,n do
    d[i,i] ← 0
for each (i,j)∈ E do
    d[i,j] ← w(i,j); p[i,j] ← i;

for k ← 1,…, n do
    if d[k,k] < 0 then
        print ''NEGATIVE CYCLE"; stop;
    for i ← 1,…, n do
        for j ← 1,…, n do
            d'[i,j] ← d[i,j];  p'[i,j] ←p[i,j]
            if d[i,k] + d[k,j] < d [i,j] then
                d'[i,j] ← d[i,k] + d[k,j]
                p'[i,j] ← p[k,j]
    d ← d';  p ← p'
```

Transitive Closure(Graph G)

```
// INIT
for i ← 1,…, n  do
      for j ← 1,…, n  do
            if (i = j  or  (i,j) ∈ E ) then
                        T[i,j] ← 1
            else T[i,j] ← 0

// Compute closure.
for k ← 1,…, n  do
        for i ← 1,…, n  do
              for  j ← 1,…, n  do
                  T'[i,j] ← (T[i,j]  or  (T[i,k]  and  T[k,j]))
        T ← T'
```

Transitive Closure For DAG(Graph G)

```
int T[1..n, 1..n] = {0,…,0}
List L ← TOPOLOGICAL_SORT(G)

for each v in reverse(L) do
        T[v,v] ← 1
        for each u ∈ Adj[v] do
             for  j ← 1,…, n  do
                   T[v,j] ← T[v,j] or T[u,j]
```

KRUSKAL(Graph G, Weight w)

```
Forest  F ← ∅                      //Empty forest
DisjointSets  S;
List L ← SORT(E)                   //Sort edges by weight

for each v∈V do
       S.MakeSet(v)    //Enter v into structure

for each (u,v) ∈ L do
             u' ← S.Find(u)
             v' ← S.Find(v)
             if  u' ≠ v'  then
                         F ← F ∪ { (u,v) }
                         S.Union(u',v')

return F
```

PRIM(Graph G, Weight w)

```
PriorityQueue  Q
VertexSet S ← ∅

min[v0] ← 0                          //INIT
p[v0] ← null
for each vertex v ≠ v0  do
            min[v] ← ∞
            p[v] ← null

Q.Build(V,min)              // Build Priority Queue

while  Q ≠ ∅  do                 // Grow Tree
       u ← Q.DeleteMin()
       S ← S ∪ {u}
       for each v∈Adj[u]  do
           if  v ∉ S  and w(u,v) < min(v) then
                      min[v] ← w(u,v)
                      p[v] ← u
                      Q.DecreaseKey(v,min[v])

return p
```

פורד-פלקרסון:

אתחול:

זרימה 0 בכל הצלעות: $f(u,v) = f(v,u) = 0$

הגרף השיורי G_f שווה לגרף המקורי G: $c_f(u,v) = c(u,v)$

כל עוד יש מסלול (משפר) מ-s ל-t בגרף G_f:

נמצא מסלול משפר P בגרף G_f

נחשב את קיבולו השיורי $c_f(P)$

נגדיל את הזרימה ב-G לאורך P ב- $c_f(P)$:

לכל צלע (u,v) במסלול P נעדכן:

$f(u,v) = f(u,v) + c_f(P)$

$f(v,u) = -f(u,v)$

נעדכן את הגרף השיורי לאורך המסלול P:

לכל צלע (u,v) במסלול P נעדכן:

$c_f(u,v) = c(u,v) - f(u,v)$

$c_f(v,u) = c(v,u) - f(v,u)$

נחזיר את f

אלגוריתם למציאת זיווג מרבי בגרף דו-צדדי:

1. נבנה רשת זרימה N באופן הבא:
   - נוסיף שני קדקודים חדשים s,t.
   - נוסיף קשתות מכוונות מ- s לכל הקדקודים בקבוצה L.
   - נוסיף קשתות מכוונות מכל קדקוד בקבוצה R ל- t.
   - נכוון כל צלע מ- L ל- R.
   - הקיבולים של כל הקשתות יהיו 1.
2. נמצא זרימה מקסימאלית f ברשת החדשה בעזרת אלגוריתם פורד-פלקרסון.
3. הזיווג M יהיה: $(u,v)\in M$ אם ורק אם $u\in L, v\in R$ וגם $f(u,v) > 0$.

Build a string-matching automaton in O(|Σ|×m) time

COMPUTE-δ(String p[1...m])

```
δ[0,p[1]] ← 1
for each a∈Σ, a≠p[1] do
    δ[0,a] ← 0

state ← 0
for i ← 1,...,m-1 do
    δ[i,p[i+1]] ← i+1
    for each a∈Σ, a≠p[i+1] do
        δ[i,a] ← δ[state,a]
    state ← δ[state,p[i+1]]

for each a∈Σ do
        δ[m,a] ← δ[state,a]
```

KMP algorithm

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])
```

δ(State i, Character a) // Transition Function

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

// run automaton on T

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
```