①

How would you select a meeting location from one of your k favorite hang-out spots(also graph vertices) known to all of you?

Which algorithm would you use and How is this done most efficiently?

Algoritm:

Input: the edge's set S

The number of cities, size

You and your two friends' cities: a,b,c

Output: the meet city is v

```
1:n←size    //Size is the number of cities
2:dist[N][N];
3:for  i=1 to i=n do
4:   For j=1 to j=n do
5:    dist[i][j]←maxvalue;   //have no roads,maxvalue is very large
6:for  each edge ∈S do
7:    dist[u][v] ←edge[u][v];
8:for k=1 to k=n do
9:    for i=1 to i=n do
10:        for j=1 to j=n do
11:            if dist[i][j]>dist[i][k]+dist[k][j]  then
12:            dist[i][j] ←dist[i][k]+dist[k][j]
13:mintime←maxvalue;
14:v←0
15:for i=1 to i=n do
16:    time←max(dist[a][i],dist[b][i],dist[c][i]);  //you and your two friends's city is a,b,c;find meet city is v;
17:    if  time<mintime then
18:       mintime←time;
19:       v←i;
 20:return v
```

First we use Floyd to calculate the minimum time for each city;

Second ,we can test every city to find the meet city

The Time complexity is $O(n^3)$

②n standards the number of nodes

If n=2

The full binary tree have two nodes,

The well-formed bracket is ()

There is a bijection between full binary tree T and the well-formed bracket S

Suppose when n=k,there is a bijection between full binary tree and the well-formed bracket

So when n=k+2，when the added two nodes' location of T is fixed(it can be the children)

We can only add () to the fixed location to the S，so there is a bijection between full binary tree T and the well-formed bracket S

So we can get,it is a bijection

③

$$X = \{x_1, x_2 \ldots . x_n\} \quad Y = \{y_1, y_2, \ldots \ldots y_n\},$$

$$if\ Z = \{z_1, z_2 \ldots . z_k\}, if\ exist\ \{i_1, i_2 \ldots . i_k\}\ with\ i_1 \le i_2 \le \ldots i_k\ \ and\ \ x_i = z_k$$

$$then\ Z\ is\ \ the\ subsequence\ of\ \ string\ X$$

$$the\ longest\ common\ subsequence\ of\ X\ and\ Y\ is: there\ is\ a\ string\ set\ \ S, each\ s$$

$$\in S, and\ \ s\ \ is\ the\ common\ subsequence\ of\ X\ and\ Y.$$

$$we\ find\ the\ longest\ string\ Z\ in\ S$$

Example:

X = "abcde", Y = "ace" ,Z= "ace"

brute force algorithm:

Algoritm:

Input: the string X,Y

Output: string z

1:l←min(length(X), length(Y))

2: for i=l to i=1 do

3: find the set S1 that store all subsequences of X and the sequence's length is i

4: find the set S2 that store all subsequences of Y and the sequence's length is i

5: for $each\ x \in S1$ do

6: for $each\ y \in S2$ do

7: if x==y then

8: z←x

9: return z

10: S1,S2←$\emptyset$

$$the\ space\ \text{complexity is: X, Y has } 2^l\ \ string, SO\ it\ is\ O(2^l)$$
$$compare\ string\ \ a\ \ between\ string\ b\ needs\ compare\ l\ times,$$

$$\sum_{i=1}^{n} 2^i * 2^i * i = (n-1)4^{n+1} - 4$$

The time complexity is $O(n4^n)$

Prove: first we find the minimum length of X and Y is l

then we find all subsequences of X and Y,and the length of subsequence is smaller than l

for l to 1,we can find the common subsequences of X and Y,then the subsequence which is first found is the longest.

Describe a dynamic programming algorithm for the problem.

Algoritm:

Input: the string X,Y

Output: string z

```
1:N←max(length(X),length(Y))
2:for i=0 to i=N do
3:  for j=0 to j=N do
4:          d[i][j] ←0
5:          dir[i][j] ←-1
6:for i=1 to i=N do
7:  for j=1 to j=N do
8:          if X[i-1]==Y[j-1] then
9:                  d[i][j] ←d[i-1][j-1]+1
10:                         dir[i][j] ←1
11:                 else if d[i][j-1]>d[i][j] then
12:                         d[i][j] ←d[i][j-1]
13:                         dir[i][j] ←2
14:                 else then
15:                         d[i][j] ←d[i-1][j]
16:                         dir[i][j] ←3
17:i←length(X)
18:j←length(Y),lcs←””
19:while !((i==0)||(j==0)) do
20:                 if X[i]==Y[j] then
   21:                         z←z||X[i]
22:                 if dir[i][j]==1 then
23:                         i←i-1
```

```
24:                 j←j-1
25:             if dir[i][j]==2 then
26:                 j←j-1
27:             if dir[i][j]==3 then
28:                 i←i-1
29:             return z
```

The time complexity is $O(n^2)$

Illustrate an example of reasonable size how the algorithm works.

X=”abcde” Y=”acde”

First step

	a	c	d	e
a	0	0	0	0
b	0	0	0	0
c	0	0	0	0
d	0	0	0	0
e	0	0	0	0

Second step

	a	c	d	e
a	1	1	1	1
b	0	0	0	0
c	0	0	0	0
d	0	0	0	0
e	0	0	0	0

Third step

	a	c	d	e
a	1	1	1	1
b	1	1	1	1
c	1	2	2	2
d	0	0	0	0
e	0	0	0	0

…..

We can get

	a	c	d	e
a	1	1	1	1
b	1	1	1	1
c	1	2	2	2
d	1	2	3	3
e	1	2	3	4

We can get the length is 4

Dir[i][j]=1 stands the x[i]=x[j]

So we can use dir to determine the direction

Dir[i][j]=2 stands the common subsequence choose the direction of y

Dir[i][j]=3 stands the common subsequence choose the direction of x

So we use the while loop to find z

(4)

$n = 6$ $P = 3, 10, 2, 12, 5, 50, 4$

$P_0 = 3$ $P_1 = 10$ $P_2 = 2$ $P_3 = 12$ $P_4 = 5$ $P_5 = 50$
$P_6 = 4$

$A_1 = 3 \times 10 = 30$ $A_2 = 10 \times 2 = 20$ $A_3 = 2 \times 12 = 24$
$A_4 = 12 \times 5 = 60$ $A_5 = 5 \times 50 = 250$ $A_6 = 50 \times 4 = 200$

$r = 1$ $m[1,1] = 0$ $m[2,2] = 0$ ... $m[6,6] = 0$

$r = 2$ $m[1,2] = 3 \times 10 \times 2 = 60$

$m[2,3] = 10 \times 2 \times 12 = 240$

$m[3,4] = 2 \times 12 \times 5 = 120$

$m[4,5] = 12 \times 5 \times 50 = 3000$

$m[5,6] = 5 \times 50 \times 4 = 1000$

$r = 3$: $m[1,3] = \min(m[1,2] + m[3,3] + 3 \times 2 \times 12, (m[1,1] + m[2,3] + P_0 P_1 P_3) = \min(60 + 72, 240 + 360) = 132$
($3 \times 10 \times 12$)

$m[2,4] = \min\{m[2,2] + m[3,4] + P_1 P_2 P_4, m[2,3] + m[4,4] + P_1 P_3 P_4\} = \min\{120 + 10 \times 2 \times 5, 240 + 10 \times 12 \times 5\} = 220$

Powered by CamScanner

$m[3,5) = \min\{m(3,3] + M[4,5) + P_2P_3P_5,$

$m[3,4] = \min\{3000 + 2\times15\times50, 120 + 2\times5\times50\}$

$= 220$

$m[3,5] = \min\{m[3,3) + m[4,5] + P_2P_3P_5,$

$m[3,4) + m(5,5) + P_2P_4P_5) = \min\{3000$

$+ 2\times12\times50, 120+2\times5\times50\} = 620$

$m(2,5] = \min[m(2,2) + m(3,5) + P_1P_2P_5, m(2,3)$

$+ m[4,5) + P_1P_3P_5, m[2,4] + m(5,5] + P_1P_4P_5]$

$= \min[620+10\times2\times50, 240+3000+10\times12\times50, 220+$

$10\times5\times50] = 1620$

$m[1,6) = \min[m(1,1] + m(2,6) + P_0P_1P_6,$

$m[1,2) + m(3,6) + P_0P_2P_6,$

$m[1,3) + m[4,6] + P_0P_3P_6$

$m[1,4) + m(5,6) + P_0P_4P_6$

$m[1,5) + m(6,6) + P_0P_5P_6]$

$= 1104$

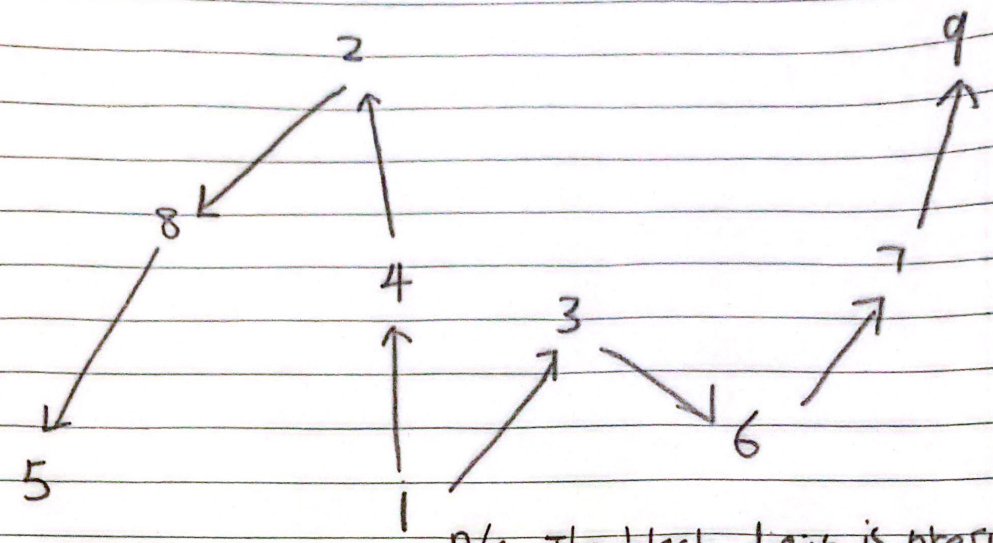
Powered by CamScanner

	0	60	132	210	960	1104
min	m[1,1]	m[1,2]	m[1,3]	m(1,4)	m(1,5]	m(1,6)
		0	240	220	1620	1100
		m[2,2]	m[2,3]	m(2,4)	m[2,5]	m(2,6)
			0	120	620	1020
			m(3,3)	m(3,4)	m(3,5)	m(3,6)
				0	3000	1240
				m(4,4)	m(4,5)	m(4,6)
					0	1000
					m(5,5)	m(5,6)
						0
						m(6,6)

Powered by CamScanner

(5)

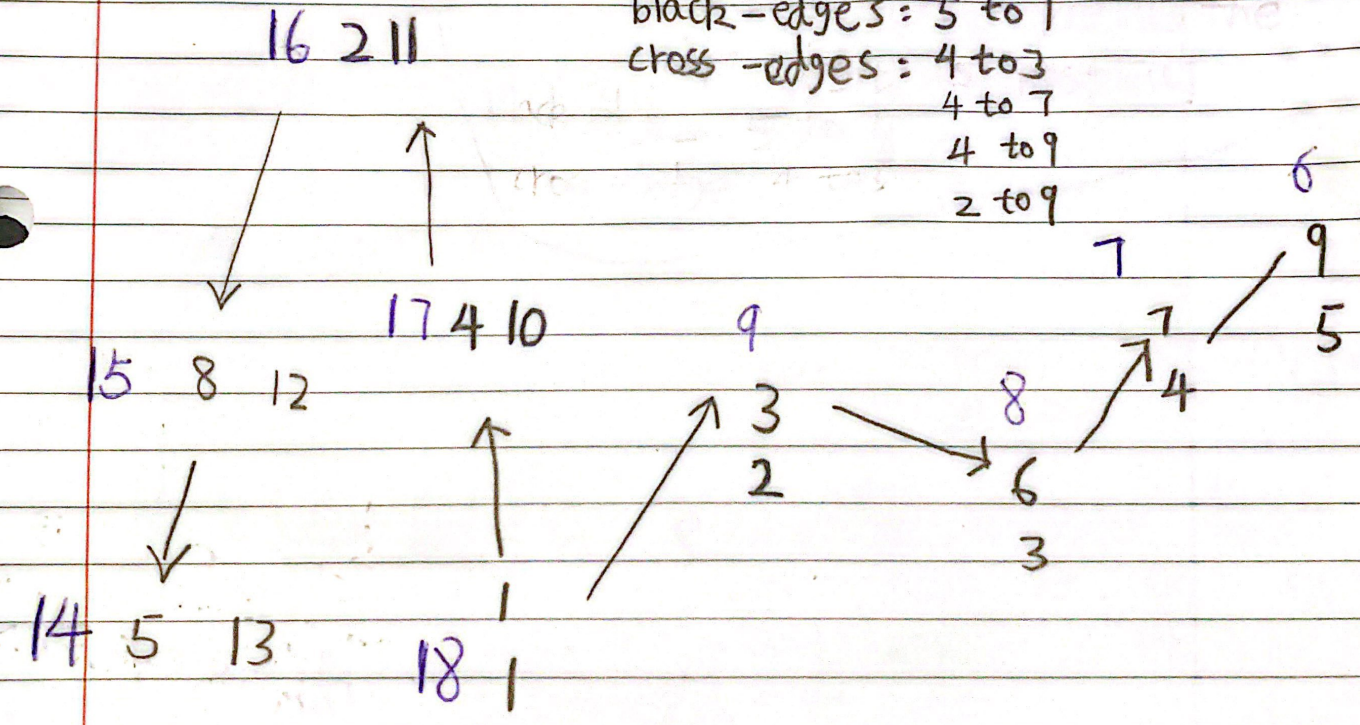
Dfs 1,3,6,7,9,4,2,8,5

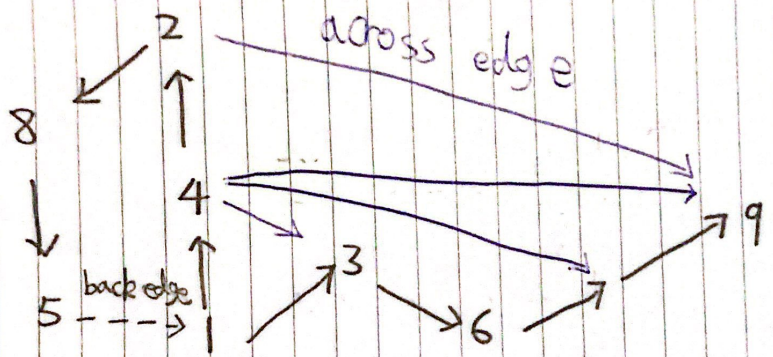


Dfs The black digit is pre[u], the blue digit is post[u]

black-edges: 5 to 1

cross-edges: 4 to 3
4 to 7
4 to 9
2 to 9



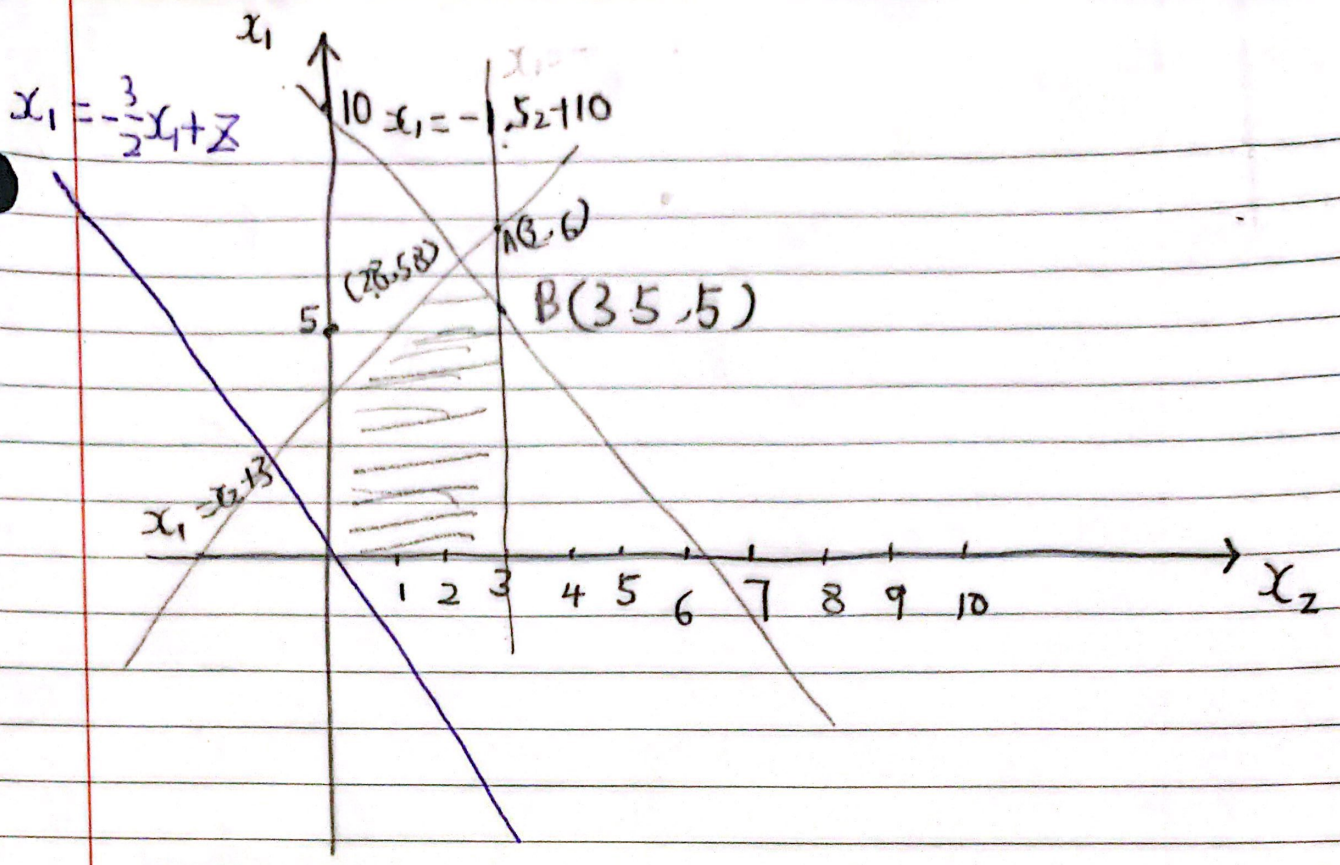


There has a road to vist all nodes and that we only visit every node once along road

We can find a path from the graph that traverses each node only once until all nodes have been traversed.

Powered by CamScanner

⑥



The shadow is feasible region

We can use line $x_1 = -1.5x_2 + \frac{z}{2}$ to find lowest z

we move the line to left until it through (0,0)

we can get $z=0$ is the lowest value of $2x_1+3x_2$

It's the only solution

Because when $x_1 = -1.5x_2 + \frac{z}{2}$ through a point (0,0)

The line is unique, and no other point which is shadow is on the line

The (0,0) is unique point which makes $2x_1+3x_2$ be lowest

Powered by CamScanner