هوالعلیم

مریم سعیدمهر
۹۶۲۹۳۷۳

دانشگاه صنعتی اصفهان ، دانشکده برق و کامپیوتر

طراحی الگوریتم ها – **تکلیف سری آخر**

موعد تحویل : ۲۵ خرداد

قسمت اول

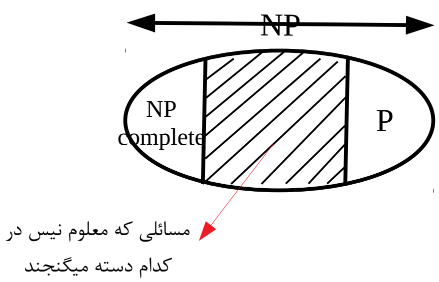
پاسخ به سوالات چند گزینه ای و توضیح درباره درستی یا غلطی هر مورد ::

۱- فرض کنید که P!=NP

(A) NP-complete = NP
(B) NP-complete ∩ P = ∅
(C) NP-hard = NP
(D) P = NP-complete

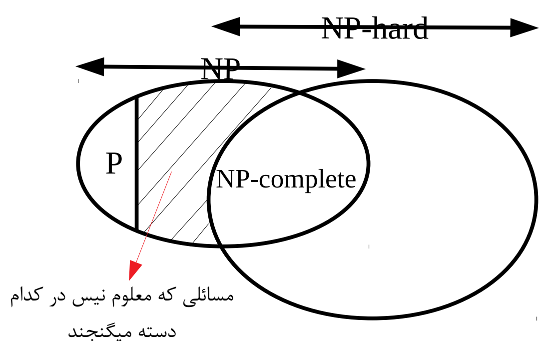
پاسخ :

(A) : غلط است ، زیرا



(B) : درست است زیرا اگر اشتراک داشته باشند تمام مسائل موجود در دسته ی NP ها حل میشوند(درزمان چندجمله ای) و در نتیجه P =NP که خلاف فرض اولیه است!

(C) : غلط است ، زیرا



(D) : غلط است ، زیرا اگر این دو دسته برابر باشند کل مسائلی که در NPها میگنجند قابل حل شده(درزمان چندجمله ای) و عملاً P =NP که خلاف فرض اولیه است!

۲- فرض کنید S یک مسئله NP-complete است و Q و R دو مسئله هستند که NP نیستند.مسئله ی Q را میتوان به مسئله ی S کاهش دهیم(در زمان چندجمله ای) و همچنین مسئله S را میتوانیم کاهش دهیم به مسئله ی R (در زمان چندجمله ای)

کدام درست است؟

(A) R عضو NP-complete

(B) R عضو NP-hard

C) Q عضو NP-complete

D) Q عضو NP-hard

پاسخ : گزینه ی (B) درست است. زیرا :

در وهله ی اول دو گزینه ی (A)و (C) حذف میشوند به این دلیل که در صورت سوال گفته شده« Q و R دو مسئله هستند که NP نیستند» پس نمیتوانند در NP-complete هم قرار گیرند چرا که این دسته هم زیرمجموعه ای از NP است. گزینه ی(B) درست است زیرا مسئله S که NP-complete است به مسئله R کاهش پیدا کرده و این عمل وقتی امکان پذیر است که هر دو در یک گروه از مسائل باشند حالا که R در NP قرار ندارد پس حتما در NP-hard است که این کاهش پذیری صورت گرفته! گزینه ی(D) دقیقاً به دلیل نقیض دلیل مطرح شده در قبل، غلط است.

۳- فرض کنید X یک مسئله متعلق به NP باشد. کدام درست است؟

الف) برای X هیچ الگوریتم چند جمله ای وجود ندارد.

ب) اگر X بتواند در زمان چندجمله ای به صورت deterministic حل شود آنگاه P=NP است.

ج) اگر X عضو NP-hard باشد آنگاه X یک مسئله ی NP-complete است.

د) X شاید غیرقابل تصمیم گیری باشد(undecidable)

پاسخ : گزینه ی(ج) درست است! زیرا اگر X متعلق به دسته ی p باشد پس حتماً الگوریتم چندجمله ای دارد(گزینه ی الف غلط است) به علاوه اگر X متعلق به دسته ی NP-complete باشد و یک حل deterministic برایش پیدا کنیم آنگاه P=NP ولی اگر X متعلق به دسته ی p باشد به صورت اتوماتیک حل deterministic دارد(گزینه ی ب لزوماً درست نیست). گزینه ی د هم دیگه واضحه غلطه :/

۴- در ارتباط با مسائل Sat-3 و Sat-2

الف) هر دو در p هستند.

ب) هر دو NP-complete هستند.

ج) مسئله ی Sat-3 یک مسئله NP-complete است و Sat-2 در p است.

د) گزینه ی (الف) و (ب)

پاسخ : گزینه ی (ج) درست است!این موضوع در کلاس مطرح شده ! (حوصله ی شرح داستان نیست :( )

۵- کدام درست است؟

جمله اول) مسئله ی وجود دور در یک گراف بدون جهت یک مسئله P است.

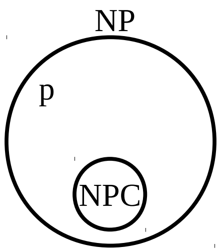
جمله دوم) مسئله ی جمله ی قبل یک مسئله ی NP است.

جمله سوم) اگر یک مسئله NP-complete باشد برای آن یک الگوریتم non-deterministic وجود دارد که در زمان چندجمله ای آن مسئله راحل میکند.

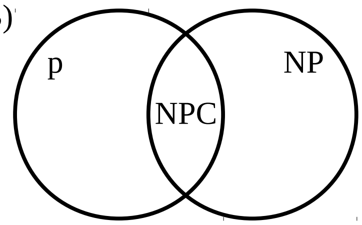
پاسخ : جملات اول و دوم و سوم درست هستند! زیرا مسئله ی وجود دور در یک گراف بدون جهت با یک DFS یا BFS در زمان چندجمله ای O(V+E) قابل حل است(جمله اول درست است) به علاوه از آنجایی که $PN \sqsupseteq$ پس مسئله ی وجود دور در یک گراف بدون جهت یک مسئله NP هم هست(جمله دوم درست است) همچنین از آنجایی که $PN \sqsupseteq$ $Ipmoc$ و NP=**N**on-deterministic **P**olynomial (جمله سوم درست است)

۶- اگر برای مسئله Clique در یک گراف حل چندجمله ای پیدا شود آنگاه کدام درست است ؟

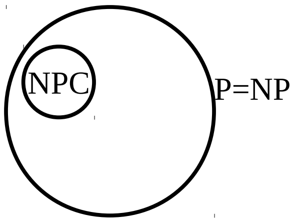
(A)



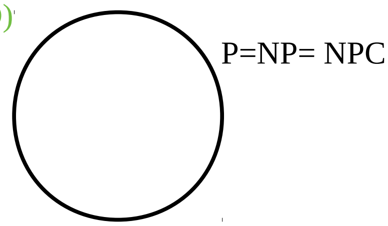
(B)



(C)



(D)



پاسخ : گزینه ی (D) درست است! زیرا مسئله ی Clique یک مسئله ی NP-complete است و اگر برای آن حل چندجمله ای پیدا شود تمام مسائل موجود در دسته ی NPها حل میشوند که در این صورت یک مجموعه ی یکپارچه خواهیم داشت و عملاً دسته بندی هایی نظیر P و NPC وجود نخواهد داشت!

قسمت دوم

۱- در قسمت الگوریتم های حریصانه با مسئله رنگ آمیزی و الگوریتمی برای رنگ آمیزی گراف ها با دو رنگ آشنا شدیم. حال فرض کنید که درهمان مسئله میخواهیم گراف را با m رنگ، رنگ آمیزی کنیم(برای سادگی کار فرض کنید m=3 است) الگوریتمی ارائه دهید که گراف مورد نظر را بررسی کند و بگوید که آیا این گراف را میتوان با m رنگ ، رنگ آمیزی کرد یا خیر. الگوریتم خود را از نظر زمان بررسی کنید(بررسی از نظر worst-case کافیست)

پاسخ :

```
# https://www.geeksforgeeks.org/m-coloring-problem-backtracking-5/  :)))
# Python program for solution of M Coloring
# problem using backtracking
class Graph():
	def __init__(self, vertices):
		self.V = vertices
		self.graph = [[0 for column in range(vertices)] for row in range(vertices)]

	# A utility function to check if the current color assignment
	# is safe for vertex v
	def isSafe(self, v, colour, c):
		for i in range(self.V):
			if self.graph[v][i] == 1 and colour[i] == c:
				return False
		return True

	# A recursive utility function to solve m
	# coloring problem
	def graphColourUtil(self, m, colour, v):
		if v == self.V:
			return True

		for c in range(1, m+1):
			if self.isSafe(v, colour, c) == True:
				colour[v] = c
```

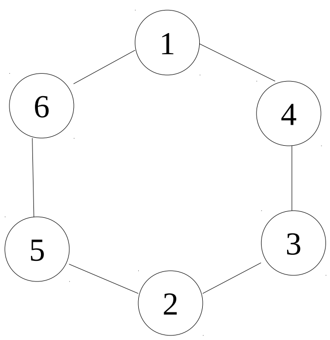
```
                if self.graphColourUtil(m, colour, v+1) == True:
                    return True
                colour[v] = 0

    def graphColouring(self, m):
        colour = [0] * self.V
        if self.graphColourUtil(m, colour, 0) == None:
            return False # means that there is not a way to color the graph
        return True # means that there is a way to color the graph
```

تحلیل زمانی : $\mathbf{O((vm)^{v})}$ که در آن v تعداد رئوس و m همان تعداد رنگ هاست .

۲- عدد طبیعی n را در نظر بگیرید. اعداد از ۱ تا n را روی یک دایره کنار هم بچینید به گونه ای که جمع هر دو عدد که کنار یکدیگر واقع شده اند یک عدد فرد بشود مثلاً برای n=6 به شکل زیر میشود :

الگوریتمی ارائه دهید که همه جواب های ممکن را برای عدد n به دست آورد.



پاسخ : اولاً حاصل جمع دو عدد در صورتی فرد است که یکی زوج و دیگری فرد باشد.

دوماً اگر n عددی فرد باشد مسئله جواب ندارد زیرا از آنجایی که اعداد باید روی دایره قرار گیرند پس عدد اول و آخر کنار یکدیگر خواهند بود و اگر n عددی فرد باشد ، عدد اول و آخر یا هردو زوج یا هر دو فرد خواهند بود که در این صورت جمعشان زوج میشود و غلط است.

ثالثاً اگر n عددی زوج باشد که مسئله جواب داشته باشد، n/۲ اعداد زوج و n/۲ مابقی فرد است و کافیست این اعداد را یکی در میان روی یک خط بچینیم و حال کافیست دو انتهای خط را کنار یکدیگر بگذاریم تا دایره مطلوب ساخته شود

در نهایت برای ساختن جواب های مختلف کافیست اعداد زوج را با هم و فردها را باهم تغییر دهیم.

```
odd = []
even = []

def toString(List):
    return ''.join(List)

def permute(a, l, r, arr):
    if l==r:
        arr.append(toString(a))
    else:
        for i in xrange(l,r+1):
            a[l], a[i] = a[i], a[l]
            permute(a, l+1, r, arr)
            a[l], a[i] = a[i], a[l] # backtrack

# Driver program to test the above function
n = int(input())
string_even = ""
string_odd = ""
for i in range(1;n+1):
    if i%2 == 0 :
        string_even.join(i)
    else :
```

```
                            string_odd.join(i)
length = len(string_odd)
a_odd = list(string_odd)
a_even = list(string_even)
permute(a_odd, 0, length-1, odd)
permute(a_even, 0, length-1, even)
for i in range(len(odd)):
  for j in range(len(even)):
    for k in range(length):
      print(odd[i][k],end=" ")
      print(even[j][k],end=" ")
    print()
```