Finals Review

CS3230 AY21/22 Sem 2

Kingston Kuan
YANG Mingyang

1. True or False?

1. True or False?

(a) Suppose the amortized cost of an operation op over 7 calls to it is 4. It is possible that for 3 out of the 7 calls, the actual cost of op is 10.

1. True or False?

(a) Suppose the amortized cost of an operation op over 7 calls to it is 4. It is possible that for 3 out of the 7 calls, the actual cost of op is 10.

Amortized Analysis

- ***Amortized analysis*** is a strategy for analyzing a sequence of operations to show that the average cost per operation is small, even though a single operation within the sequence might be expensive.

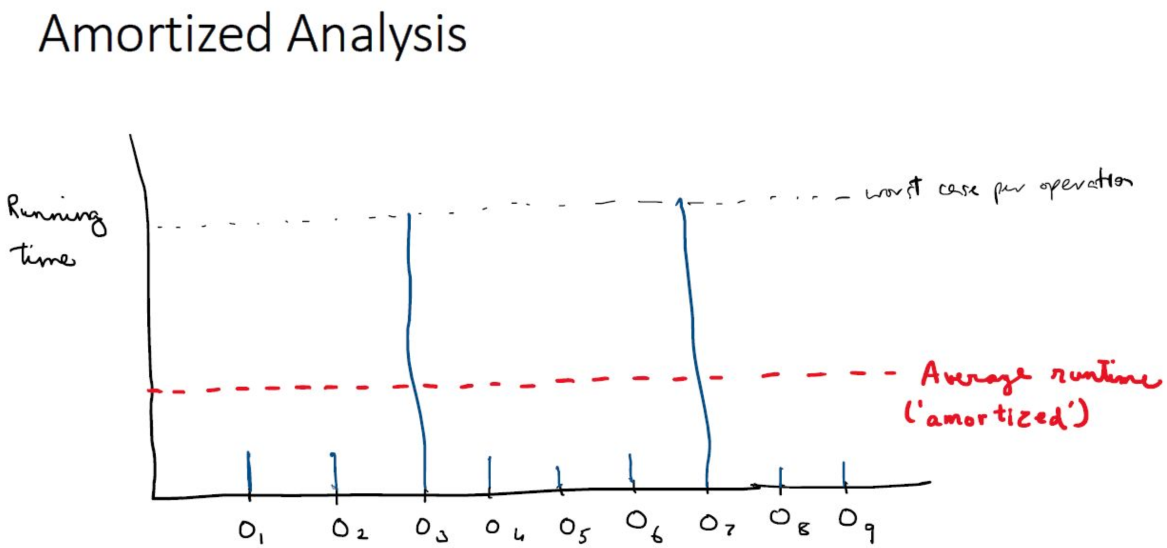
- Note, ***no probability is involved***!

Do not get confused with the average-case analysis

- An amortized analysis *guarantees* the *average performance* of each operation in the *worst case*.

1. True or False?

(a) Suppose the amortized cost of an operation op over 7 calls to it is 4. It is possible that for 3 out of the 7 calls, the actual cost of op is 10.



1. True or False?

(a) Suppose the amortized cost of an operation op over 7 calls to it is 4. It is possible that for 3 out of the 7 calls, the actual cost of op is 10.

From the amortized cost of 4, the **total cost must be 4*7 = 28**.

1. True or False?

(a) Suppose the amortized cost of an operation op over 7 calls to it is 4. It is possible that for 3 out of the 7 calls, the actual cost of op is 10.

From the amortized cost of 4, the **total cost must be 4*7 = 28**.

3 calls together already have **cost 3*10 = 30**?

1. True or False?

(a) Suppose the amortized cost of an operation op over 7 calls to it is 4. It is possible that for 3 out of the 7 calls, the actual cost of op is 10.

From the amortized cost of 4, the **total cost must be 4*7 = 28**.

3 calls together already have **cost 3*10 = 30**?

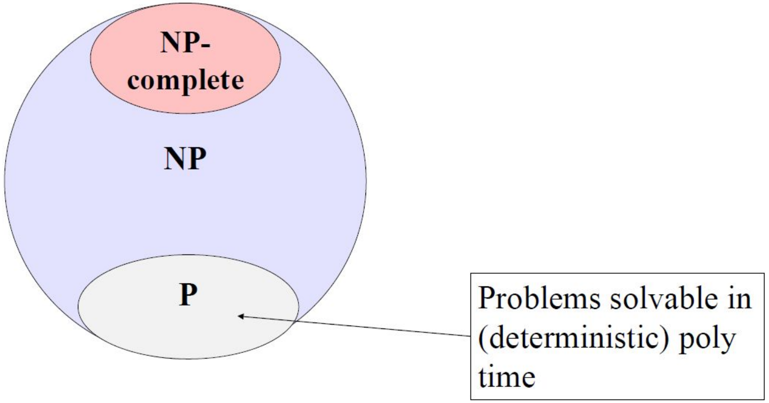
False.

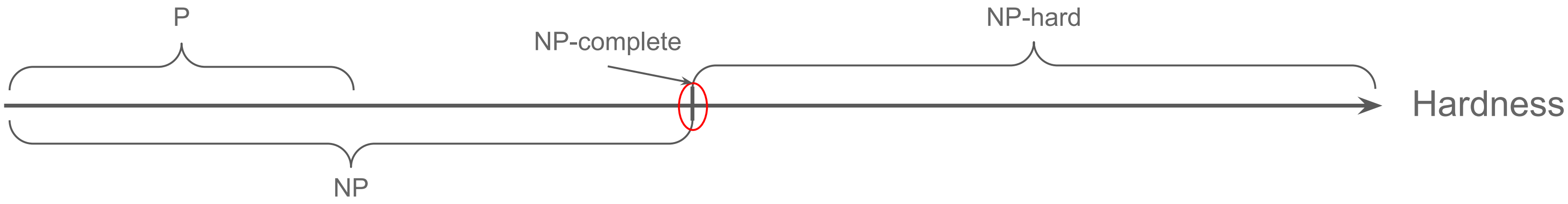
1. True or False?

(b) You learn about a problem GraphJiraffication that is in NP. You secretly work on the problem for years and finally come up with a polynomial-time algorithm. You have shown P = NP.

1. True or False?

(b) You learn about a problem GraphJiraffication that is in NP. You secretly work on the problem for years and finally come up with a polynomial-time algorithm. You have shown P = NP.

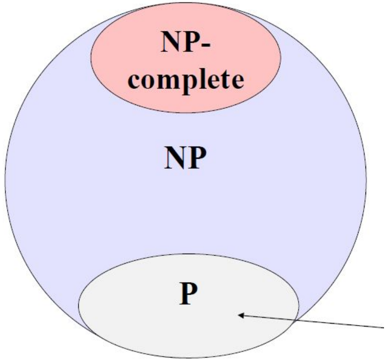




1. True or False?

(b) You learn about a problem GraphJiraffication that is in NP. You secretly work on the problem for years and finally come up with a polynomial-time algorithm. You have shown P = NP.

All **problems in P are in NP**, so you have not shown P = NP. If you could come up with a polynomial-time algorithm for an **NP-complete problem**, you would show P = NP.

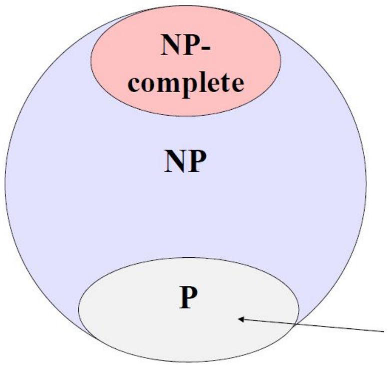


1. True or False?

(b) You learn about a problem GraphJiraffication that is in NP. You secretly work on the problem for years and finally come up with a polynomial-time algorithm. You have shown P = NP.

All **problems in P are in NP**, so you have not shown P = NP. If you could come up with a polynomial-time algorithm for an **NP-complete problem**, you would show P = NP.

False.



2. Counterexample to MIS greedy algorithm

2. Counterexample to MIS greedy algorithm

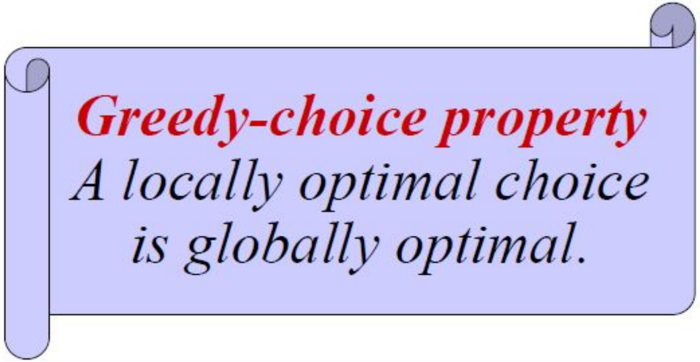
Consider the following greedy algorithm for the maximum independent set problem. Find a lowest-degree vertex v and put v in the independent set; remove v and its neighbors from the graph; repeat until no vertices remain.

Give a counterexample to show that this algorithm does not always yield a correct answer.

2. Counterexample to MIS greedy algorithm

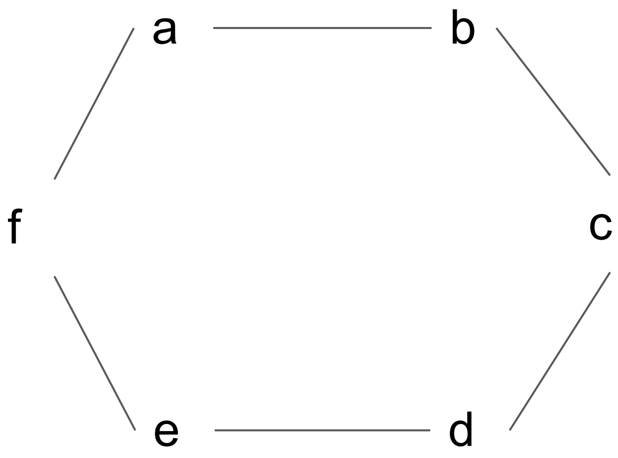
Consider the following greedy algorithm for the maximum independent set problem. Find a lowest-degree vertex v and put v in the independent set; remove v and its neighbors from the graph; repeat until no vertices remain.

Give a counterexample to show that this algorithm does not always yield a correct answer.



2. Counterexample to MIS greedy algorithm

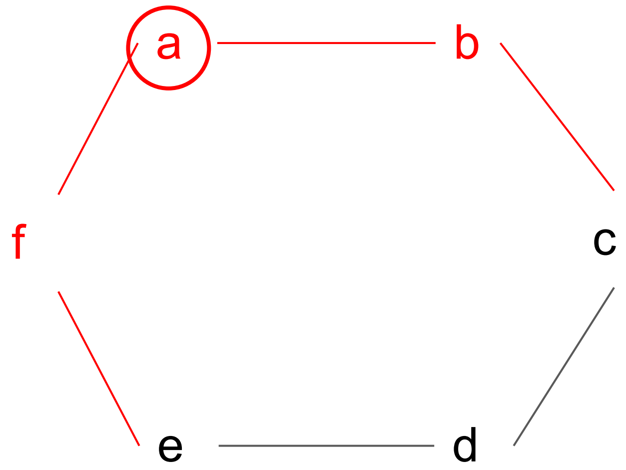
First consider a cycle on 6 nodes a, b, c, d, e, f. MIS should be size 3.



2. Counterexample to MIS greedy algorithm

“Find a lowest-degree vertex v and put v in the independent set; remove v and its neighbors from the graph; repeat until no vertices remain.”

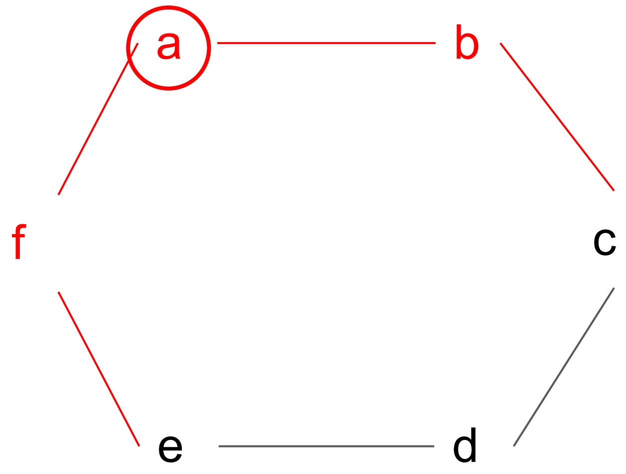
The greedy algorithm may pick a first, remove it and its neighbours.



2. Counterexample to MIS greedy algorithm

“Find a lowest-degree vertex v and put v in the independent set; remove v and its neighbors from the graph; repeat until no vertices remain.”

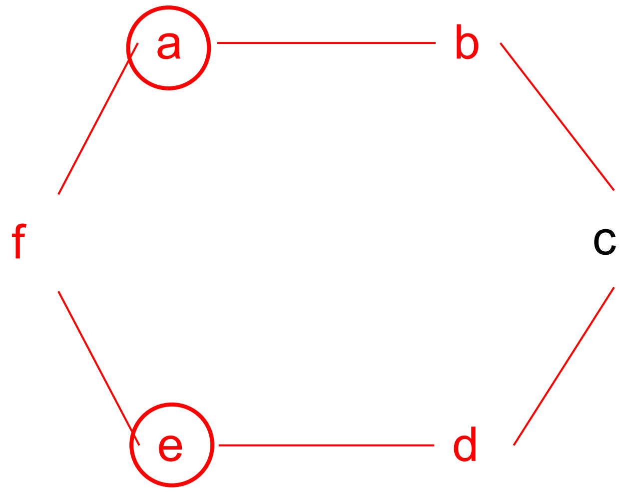
The greedy algorithm may pick a first, remove it and its neighbours.



2. Counterexample to MIS greedy algorithm

“Find a lowest-degree vertex v and put v in the independent set; remove v and its neighbors from the graph; repeat until no vertices remain.”

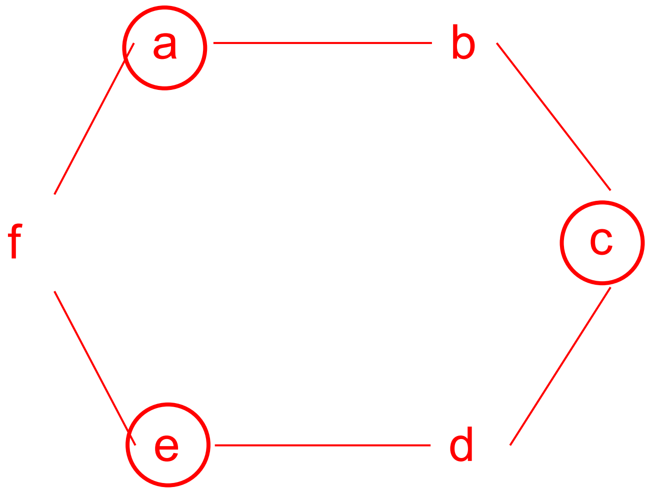
Then pick e, and only c remains



2. Counterexample to MIS greedy algorithm

"Find a lowest-degree vertex v and put v in the independent set; remove v and its neighbors from the graph; repeat until no vertices remain."

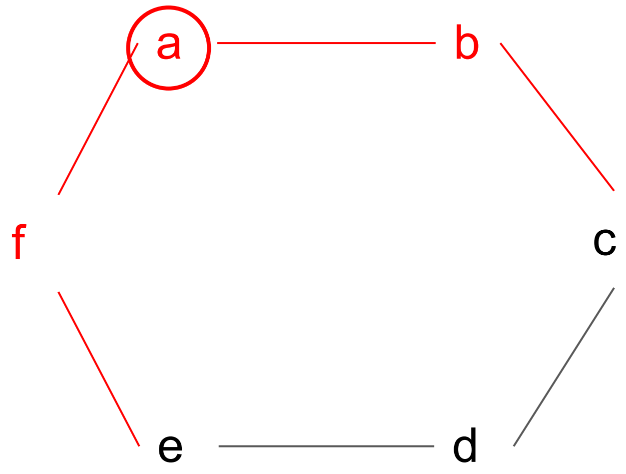
Pick c and we have found an MIS with size 3. This is correct, how can we force a wrong answer instead?



2. Counterexample to MIS greedy algorithm

“Find a lowest-degree vertex v and put v in the independent set; remove v and its neighbors from the graph; repeat until no vertices remain.”

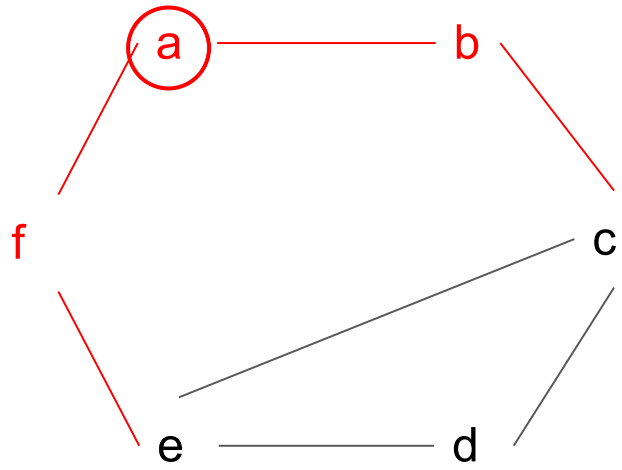
Return to the step before choosing e. We must make sure e,d,c are removed at the same time.



2. Counterexample to MIS greedy algorithm

“Find a lowest-degree vertex v and put v in the independent set; remove v and its neighbors from the graph; repeat until no vertices remain.”

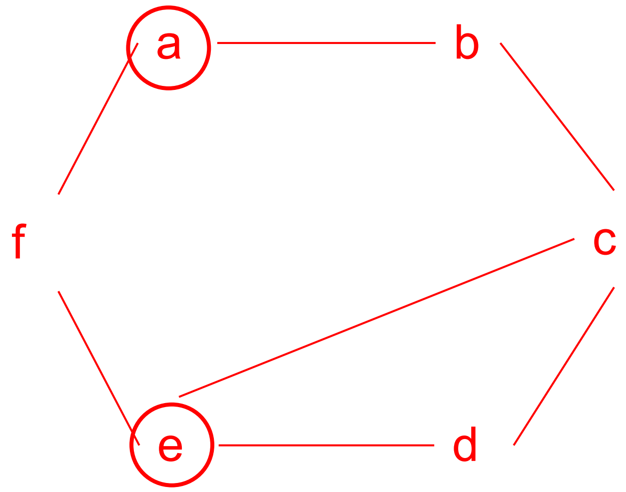
Add an additional edge between c and e. Does not affect the first choice of a.



2. Counterexample to MIS greedy algorithm

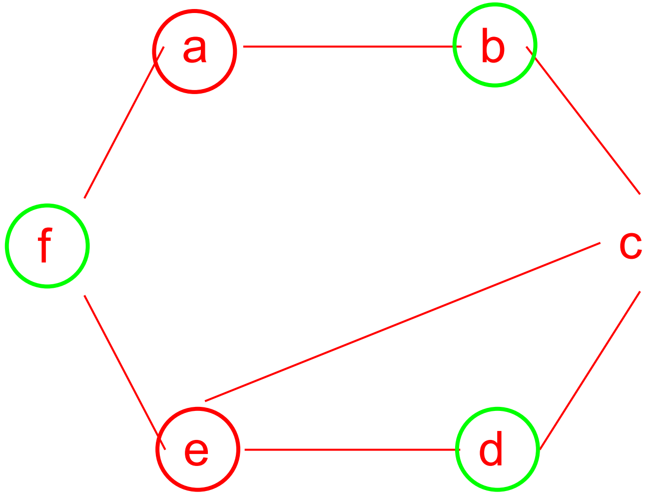
“Find a lowest-degree vertex v and put v in the independent set; remove v and its neighbors from the graph; repeat until no vertices remain.”

Then pick e, and no more vertices will remain.



2. Counterexample to MIS greedy algorithm

a and e give an independent set of **size 2**. But b, d, f forms an independent set of **size 3**.



3. Minimize difference of two sets

3. Minimize difference of two sets

Suppose you are given an array of n integers $[a_1, \ldots, a_n]$ between 0 and M. Design and analyze an algorithm for dividing the array indices into two sets X and Y such that

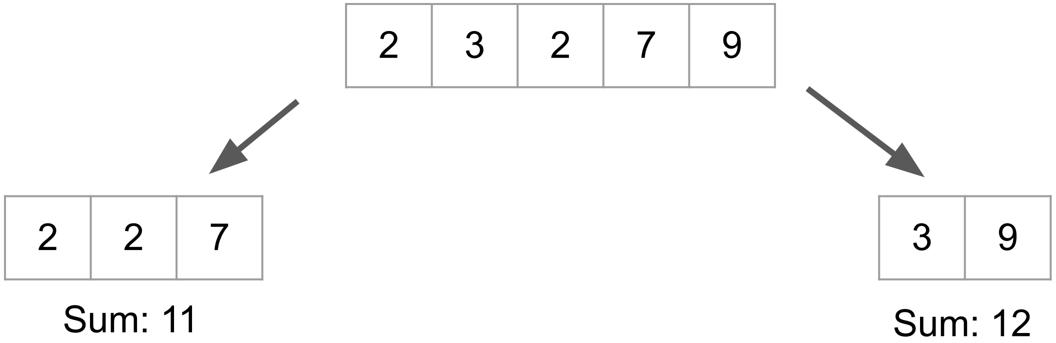
$$\left|\sum_{i \in X} a_i - \sum_{i \in Y} a_i\right|$$

,the difference of the sum of the integers in each set, is minimized.

$$\left|\sum_{i \in X} a_i - \sum_{i \in Y} a_i\right|$$

3. Minimize difference of two sets

Example: given the array [2, 3, 2, 7, 9], elements at positions 1, 3, 4 sum up to 2 + 2 + 7 = 11, while the elements at positions 2; 5 sum up to 3 + 9 = 12, yielding a difference of 1.



$$\left|\sum_{i\in X} a_i - \sum_{i\in Y} a_i\right|$$

3. Minimize difference of two sets

Let T be the sum of all the integers. Note that $T \leq nM$.

Goal: Maximize elements in one set with a limit of $\lfloor T/2 \rfloor$.

Easier to maximize one set than two at the same time.

$$\left|\sum_{i\in X} a_i - \sum_{i\in Y} a_i\right|$$

3. Minimize difference of two sets

Let T be the sum of all the integers. Note that T ≤ nM.

Goal: Maximize elements in one set with a limit of $\lfloor T/2 \rfloor$.

Naive algorithm: Compute and check all $O(2^n)$ subsets of the array

$$\left|\sum_{i \in X} a_i - \sum_{i \in Y} a_i\right|$$

3. Minimize difference of two sets

Let T be the sum of all the integers. Note that T ≤ nM.

Goal: Maximize elements in one set with a limit of $\lfloor T/2 \rfloor$.

Naive algorithm: Compute and check all $O(2^n)$ subsets of the array

Are there $O(2^n)$ **subproblems**? Have we seen a similar problem?

$\left|\sum_{i \in X} a_i - \sum_{i \in Y} a_i\right|$

3. Minimize difference of two sets

Let T be the sum of all the integers. Note that T ≤ nM.

Goal: Maximize elements in one set with a limit of ⌊T/2⌋.

Recall from lectures:

Formal Definition

KNAPSACK

Input:

$(w_1, v_1), (w_2, v_2), \ldots, (w_n, v_n)$, and W

Output: A subset $S \subseteq \{1, 2, \ldots, n\}$ that maximizes

$\Sigma_{i \in S} v_i$ such that $\Sigma_{i \in S} w_i \leq W$

2^n subsets, so naïve algorithm is too costly!

$$\left|\sum_{i\in X} a_i - \sum_{i\in Y} a_i\right|$$

3. Minimize difference of two sets

Let T be the sum of all the integers. Note that T ≤ nM.

Goal: Maximize elements in one set with a limit of ⌊T/2⌋.

Invoke the **DP algorithm for knapsack** with the given array as the weights and values, and the total knapsack weight limit being ⌊T/2⌋.

Note: “Reduction” used here but not required to prove algorithm correctness (“**Design** and **analyze** an algorithm”). Read the questions carefully, don’t spend too long on one question!

$\left|\sum_{i \in X} a_i - \sum_{i \in Y} a_i\right|$

3. Minimize difference of two sets

Let T be the sum of all the integers. Note that T ≤ nM.

Goal: Maximize elements in one set with a limit of ⌊T/2⌋.

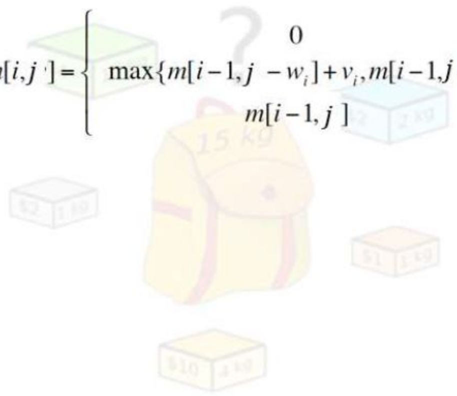
Invoke the **DP algorithm for knapsack** with the given array as the weights and values, and the total knapsack weight limit being ⌊T/2⌋.

Note: “Reduction” used here but not required to prove algorithm correctness (“**Design** and **analyze** an algorithm”). Read the questions carefully, don’t spend too long on one question!

How to analyze DP?

3. Minimize difference of two sets

How to analyze DP?



$$m[i,j] = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ \max\{m[i-1,j-w_i]+v_i,\, m[i-1,j]\} & \text{if } w_i \le j \\ m[i-1,j] & \text{otherwise} \end{cases}$$

$v[1..5] = \{4, 2, 10, 1, 2\}$

$w[1..5] = \{12, 1, 4, 1, 2\}$

$W = 15$

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	j
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	1	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4	
i	**2**	0	2	2	2	2	2	2	2	2	2	2	2	4	6	6	6	
	3	0	2	2	2	10	12	12	12	12	12	12	12	12	12	12	12	
	4	0	2	3	3	10	12	13	13	13	13	13	13	13	13	13	13	
	5	0	2	3	4	10	12	13	14	15	15	15	15	15	15	15	**15**	

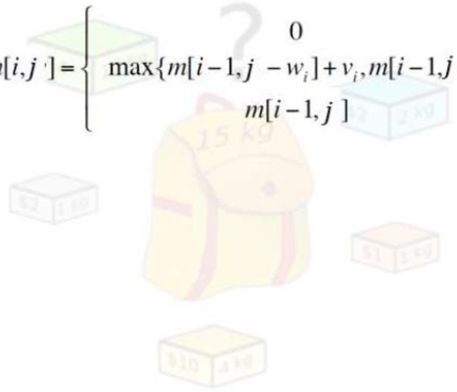
3. Minimize difference of two sets

How to analyze DP?

Time per table entry = O(1)

Table size = O(nW)

Total time = O(nW)



$$m[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ \max\{m[i-1, j - w_i] + v_i, m[i-1, j]\} & \text{if } w_i \le j \\ m[i-1, j] & \text{otherwise} \end{cases}$$

$v[1..5] = \{4, 2, 10, 1, 2\}$

$w[1..5] = \{12, 1, 4, 1, 2\}$

$W = 15$

i \ j	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
2	0	2	2	2	2	2	2	2	2	2	2	2	4	6	6	6
3	0	2	2	2	10	12	12	12	12	12	12	12	12	12	12	12
4	0	2	3	3	10	12	13	13	13	13	13	13	13	13	13	13
5	0	2	3	4	10	12	13	14	15	15	15	15	15	15	15	**15**

$$\left|\sum_{i\in X} a_i - \sum_{i\in Y} a_i\right|$$

3. Minimize difference of two sets

Let T be the sum of all the integers. Note that $T \leq nM$.

Goal: Maximize elements in one set with a limit of $\lfloor T/2 \rfloor$.

Invoke the **DP algorithm for knapsack** with the given array as the weights and values, and the total knapsack weight limit being $\lfloor T/2 \rfloor$.

Table size is $O(n * \lfloor T/2 \rfloor) = O(n^2M)$, time per table entry is $O(1)$.

The algorithm runs in time $O(n^2M)$.

$$\left|\sum_{i \in X} a_i - \sum_{i \in Y} a_i\right|$$

3. Minimize difference of two sets

Can we do better?

You should be able to do a reduction from Partition problem to see that the decision version of this problem is NP-complete (try it out!) so there is no known polynomial time solution to this problem

4. Consider the activity selection problem: You are given a set of n activities with starting times $s_1, \ldots, s_n$ and finishing times $f_1, \ldots f_n$ where each $s_i \leq f_i$. The goal is to find the largest subset of activities which don't *conflict*, meaning that for any pair of activities selected, the finishing time of one of them is not later than the starting time of the other. Consider each of the following greedy strategies. If it works, give a proof of the greedy-choice property. If it doesn't, show a counterexample.

   (a) Choose an activity that ends first, discard all that conflict with it, and recurse.

   (b) If no activities conflict, choose them all. Otherwise, discard an activity that conflicts with the most number of other activities, and recurse.

Q4

Q4). n activities

For activity i, $[s_i, f_i]$ $\forall i = 1, 2, 3, \dots, n$

$\boxed{\text{Goal:}}$ Find the largest subset of n activities.

s.t. there is no conflict / overlapping of time

Consider the Greedy Strategies.

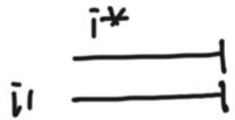
Q4

Consider the <u>Greedy Strategies.</u>

(a) Choose an activity that ends first. Discard those conflict with it. and Resurse.

$\boxed{\text{Sol}}$: Let i^* be the activity that ends first.

Given an optimal solution $i_1, i_2, \ldots, i_k$. sorted in order of the end time.

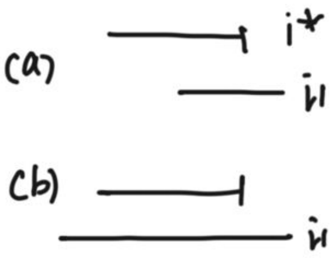


case 1: $i^* = i_1$

it gives an optimal solution by choosing the one end first.

Q4

case 2: $i^* < i_1$



Replacing i_1 with i^*.

claim: $i^*, i_2, i_3, \ldots, i_k$ is also an optimal solution

proof: since i_1 has no confrict with other activities. and i_1 is the first activity

$\because i^* < i_1$

$\therefore i^*$ has no conflict with others.

$(i^*, i_2, i_3, \ldots, i_k)$ has total k activities.

which reach the optimal number $(=k)$]

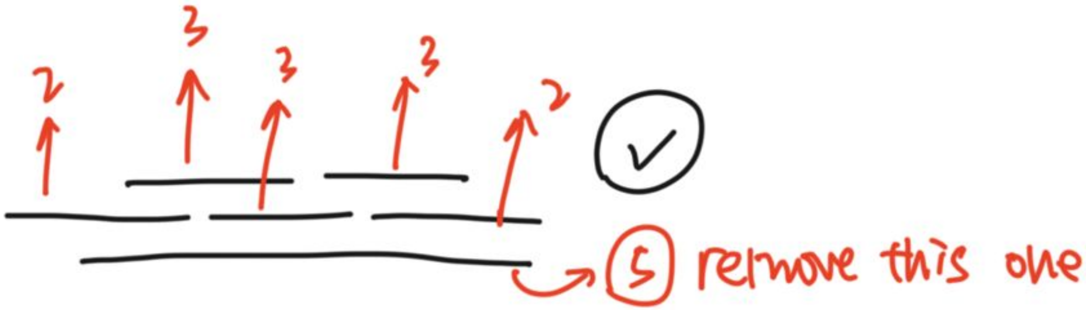
Q4

(b). No conflict → select all

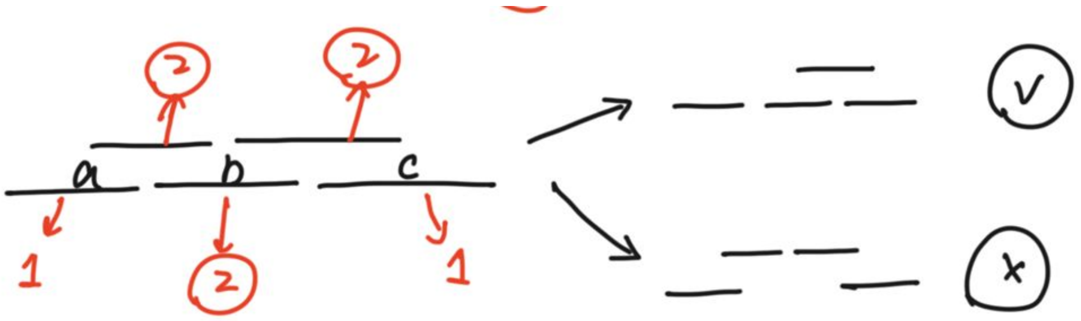
o.w. → Discard an activity that conflicts with the most # of others. Recurse.

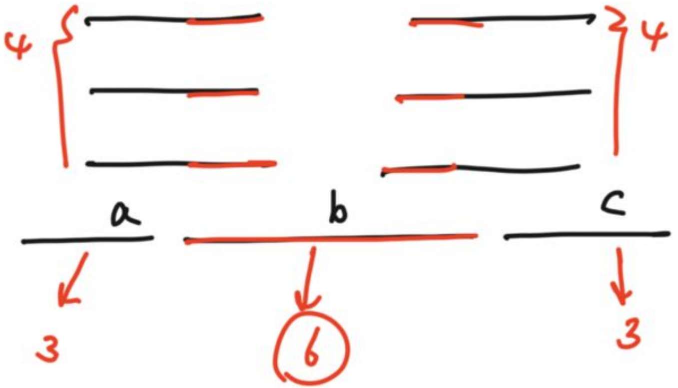
FALSE.

Intuitively,



Q4





(a) Works. Let i^* be the activity that ends first. Given an optimal solution $i_1, \ldots, i_k$ sorted in order of end time, observe that $i^*, i_2, \ldots, i_k$ is also an optimal solution. The reason is that if i_1 does not conflict with any of the other activities, i^* can't either, as it doesn't end later than i_1.

(c) Doesn't work. Suppose $s_1 = 1, f_1 = 2, s_2 = 1, f_2 = 3, s_3 = 2, f_3 = 6, s_4 = 5, f_4 = 8, s_5 = 7, f_5 = 8$. Then, activity 3 conflicts with activities 2 and 4, and two is the highest number of collisions for any activity. If activity 3 is discarded, only two out of the remaining four can be scheduled without conflicts. However, the optimal solution is of size 3: activities 1, 3, and 5.

Q5)

Define the problem PartitionEqual as follows:

Given n non-negative integers, where n is even, decide whether they can be partitioned into two parts of size n/2 each, so that both parts have the same sum.

Prove that PartitionEqual is NP-complete.

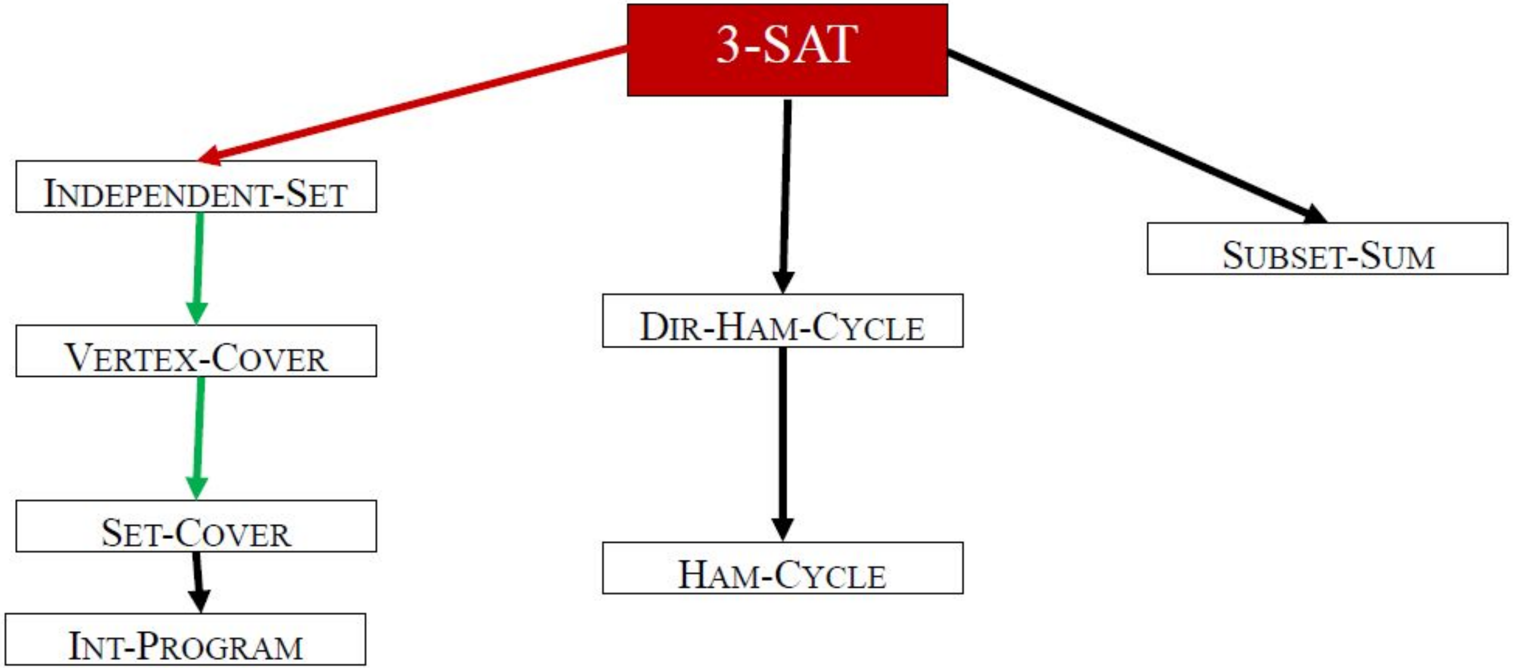
Q5)

Add one more known result that:
Partition is NP-complete

Correct the definition:
PartitionEqual: given n **non-negative** integers, where n is even, decide whether they can be partitioned into two parts of size n/2 each, so that both parts have the same sum.
Partition: given n **non-negative** integers, decide whether they can be partitioned into two parts, so that both parts have the same sum.

Reductions

NUS
National University of Singapore



Some Complements:

Given an array $A = [a_1, a_2, \ldots, a_n]$ of nonnegative integers, consider the following problems:

1 **Partition**: Determine whether there is a subset $P \subseteq [n]$ ($[n] := \{1, 2, \cdots, n\}$) such that $\sum_{i \in P} a_i = \sum_{j \in [n] \setminus P} a_j$

2 **Subset Sum**: Given some integer k, determine whether there is a subset $P \subseteq [n]$ such that $\sum_{i \in P} a_i = k$

We can show that there exists a ploynomial-time reduction from Subset Sum to Partition problem.

And we can show that Partition is NP-complete.

PartitionEqual: given n non-negative integers, where n is even, decide whether they can be partitioned into two parts of size n/2 each, so that both parts have the same sum.
Partition: given n non-negative integers, decide whether they can be partitioned into two parts, so that both parts have the same sum.

Q5)

Given that Partition is NP-complete.

Goal: prove PartitionEqual is NP-complete

PartitionEqual: given n non-negative integers, where n is even, decide whether they can be partitioned into two parts of size n/2 each, so that both parts have the same sum.
Partition: given n non-negative integers, decide whether they can be partitioned into two parts, so that both parts have the same sum.

Q5)

Given that Partition is NP-complete.

Goal: prove PartitionEqual is NP-complete

Relation between the two problems:

PartitionEqual: Partition with the extra requirement that S contain exactly n/2 elements. n is even.

PartitionEqual: given n non-negative integers, where n is even, decide whether they can be partitioned into two parts of size n/2 each, so that both parts have the same sum.
Partition: given n non-negative integers, decide whether they can be partitioned into two parts, so that both parts have the same sum.

Q5)

Given that Partition is NP-complete.

Goal: prove PartitionEqual is NP-complete

Relation between the two problems:

PartitionEqual: Partition with the extra requirement that S contain exactly n/2 elements. n is even.

Step:

1. It's easy to show that PartitionEqual is in NP, similar to Partition.

PartitionEqual: given n non-negative integers, where n is even, decide whether they can be partitioned into two parts of size n/2 each, so that both parts have the same sum.
Partition: given n non-negative integers, decide whether they can be partitioned into two parts, so that both parts have the same sum.

Q5)

Given that Partition is NP-complete.

Goal: prove PartitionEqual is NP-complete

Relation between the two problems:

PartitionEqual: Partition with the extra requirement that S contain exactly n/2 elements. n is even.

Step:

1. It's easy to show that PartitionEqual is in NP, similar to Partition.

2. Try to give a reduction from Partition to PartitionEqual.

PartitionEqual: given n non-negative integers, where n is even, decide whether they can be partitioned into two parts of size n/2 each, so that both parts have the same sum.
Partition: given n non-negative integers, decide whether they can be partitioned into two parts, so that both parts have the same sum.

Q5)

Given that Partition is NP-complete.

Goal: prove PartitionEqual is NP-complete

Relation between the two problems:

PartitionEqual: Partition with the extra requirement that S contain exactly n/2 elements. n is even.

Step:

1. It's easy to show that PartitionEqual is in NP, similar to Partition.
2. Try to give a reduction from Partition to PartitionEqual.
3. Analysis

PartitionEqual: given n non-negative integers, where n is even, decide whether they can be partitioned into two parts of size n/2 each, so that both parts have the same sum.
Partition: given n non-negative integers, decide whether they can be partitioned into two parts, so that both parts have the same sum.

Q5)

Given that Partition is NP-complete.

Goal: prove PartitionEqual is NP-complete

Step 1: show that PartitionEqual is in NP, similar to Partition.

Given n non-negative integers, sum them up and get S0.

Given n/2 selected integers, sum them up and get S1.

Check if S1 = S0/2.

The whole process runs in polynomial time.

PartitionEqual: given n non-negative integers, where n is even, decide whether they can be partitioned into two parts of size n/2 each, so that both parts have the same sum.
Partition: given n non-negative integers, decide whether they can be partitioned into two parts, so that both parts have the same sum.

Q5)

Given that Partition is NP-complete.

Goal: prove PartitionEqual is NP-complete

Step 2: construct a polynomial time reduction from Partition to PartitionEqual.

Given an instance of Partition: n numbers, [b(1), b(2), … b(n)].

Construct an instance of PartitionEqual by adding n number, with b(n+1) = b(n+2) = … = b(2n) = 0.

PartitionEqual: given n non-negative integers, where n is even, decide whether they can be partitioned into two parts of size n/2 each, so that both parts have the same sum.
Partition: given n non-negative integers, decide whether they can be partitioned into two parts, so that both parts have the same sum.

Q5)

Given that Partition is NP-complete.

Goal: prove PartitionEqual is NP-complete

Step 3: analysis

The time for construction is O(n).

Suppose a YES-instance for Partition. And denote such collection of [b(1), b(2), … b(n)] as C1 with k numbers, where $0 < k < n$.

For [b(1), b(2), … b(2n)], there exists a collection of n numbers, whose sum is ½ of the total sum by adding (n-k) zero-value integers to C1.

Therefore, it is a YES-instance for PartitionEqual.

PartitionEqual: given n non-negative integers, where n is even, decide whether they can be partitioned into two parts of size n/2 each, so that both parts have the same sum.
Partition: given n non-negative integers, decide whether they can be partitioned into two parts, so that both parts have the same sum.

Q5)

Given that Partition is NP-complete.

Goal: prove PartitionEqual is NP-complete

Step 3: analysis

Suppose a YES-instance for EqualPartition. Hence, there exists a collection C2 of [b(1), b(2), … b(2n)] with n numbers, whose sum is ½ of the total sum.

Remove n zero-value integers.

There are n non-negtive integers left among original 2n integers. The total sum remains the same.

And the remaining integers in C2 forms a new collection C1. the sum of C1 remains the same as C2. C1 is a partition of the n non-negative numbers.

Therefore, it is a YES-instance for Partition.

6. Given a directed graph G, a *simple path* is a path with no repeated vertices, while a *simple cycle* is a cycle with no repeated vertices. Consider the following two decision problems:

   - LongSimplePath: Given an unweighted **directed** graph G, two vertices u and v, and a positive integer k, decide whether there exists a simple path in G from u to v of length at least k.
   - LongSimpleCycle: Given an unweighted **directed** graph G and a positive integer ℓ, decide whether there exists a simple cycle in G on at least ℓ vertices.

   (a) Describe a polynomial-time reduction from LongSimplePath to LongSimpleCycle. Prove the correctness of your reduction.

   (b) LongSimplePath is NP-hard. From this and part (a), show that there is no known polynomial-time algorithm for LongSimpleCycle.

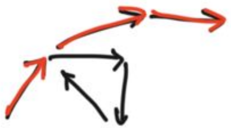
Q6

Q6 Given a directed graph. G.

a simple path: a path with no repeated vtx

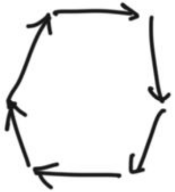
v

x

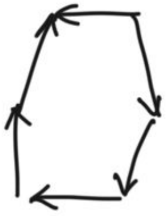


v

a simple cycle: a cycle without any repeated vtx.



v

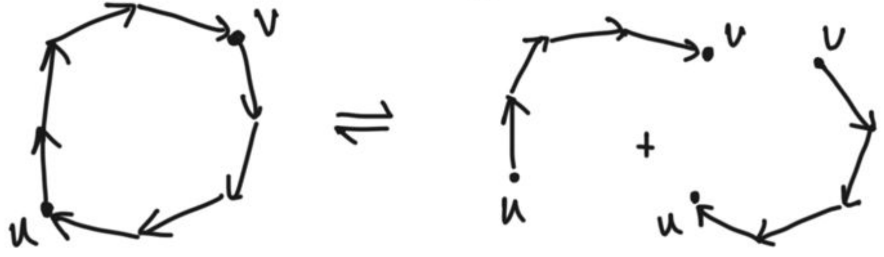


x

Q6

Relationship: from Observation

u, v are 2 vtx on the cycle C



C

$\begin{cases} u \to v \text{ path } P_1 \\ v \to u \text{ path } P_2 \\ P_1\ P_2 \text{ inner vtx - disjoint} \end{cases}$

(there is no same vtx on P_1, P_2 except u and v)

length l

length $|P_1| + |P_2| = l$

Q6

TWO DECISION PROBLEMS: Given unweighted directed graph G

LongSPath: given two vtx (u, v), $k>0$,

If there exists a simple path $u \to v$ of length $\geq k$

LongSCycle: given $l > 0$

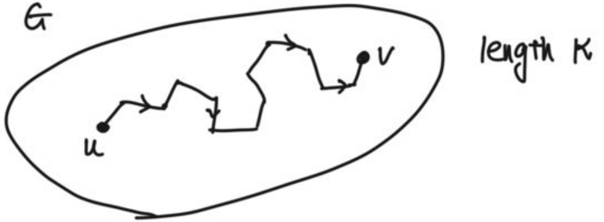
If there exists a simple cycle on at least l vtxs.

Q6

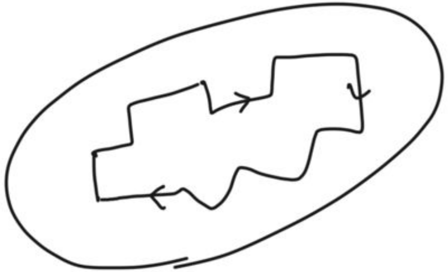
(a). Describe : LongSimplePath $\le_p$ LongSimpleCycle.
and prove it.

Intuitively: ① relationship between 2 problems

— LongSimplePath: INPUT: G. vtx u, v. integer $k > 0$



— LongSimpleCycle: INPUT: G . integer $L > 0$



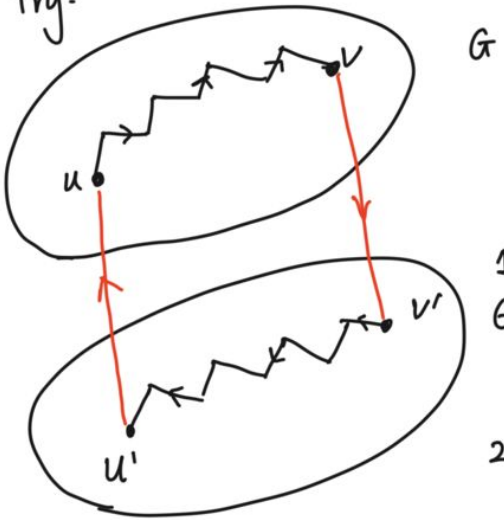
Q6

② PATH ⟶ CYCLE.

To check it exists a path → to check if exists a cycle

∴ construct a cycle based on G from a path

First try:

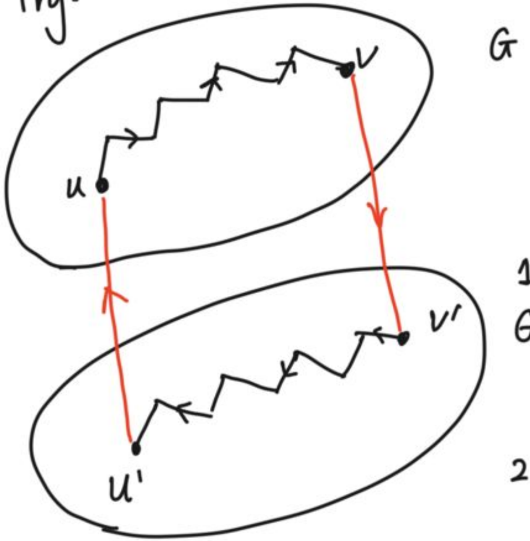


1. construct G_1
G': reverse direction of
all edges.

2. combine G and G_1 to be G'
by adding 2 edges:
$u' \to u$. $v \to v'$

Q6

First try:



1. construct G_1
G': reverse direction of all edges.
2. combine G and G_1 to be G' by adding 2 edges: $u' \to u$. $v \to v'$

The input: G'. $2k+2$

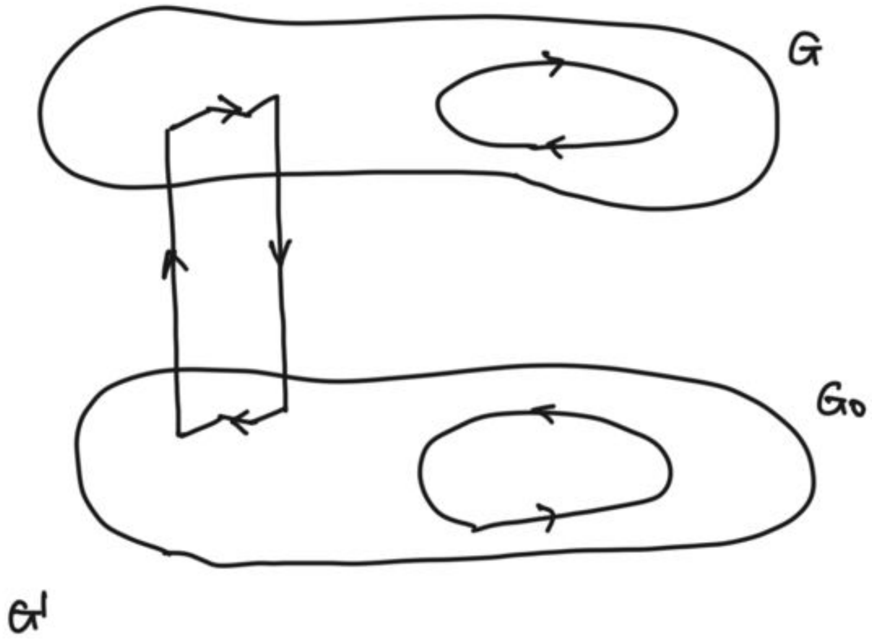
property: 1) polynomial reduction

2) YES-INSTANCE for Long Simple Path

$\Rightarrow$ YES-INSTANCE for Long Simple Cycle

Q6

However, if k is small, $2k+2$ is small
there may exist a small cycle in $G \sqsubseteq G'$



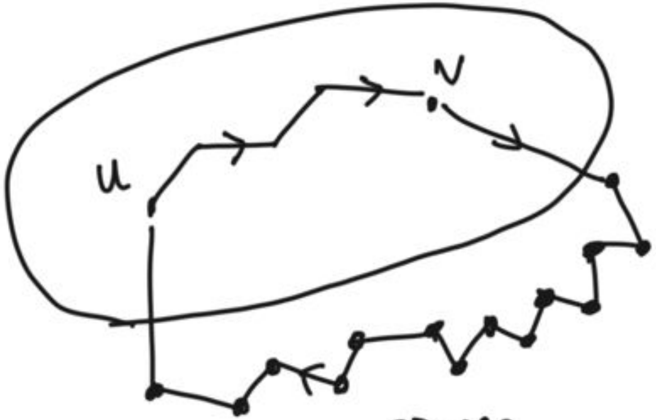
In this case:
Longsimple Cycle will always return TRUE

Q6

Add more vtx on the simple cycle of G'

↑

SECOND TRY : Avoid above situation.



simple
construct another path v → u with another n new vtxs.

① The simple cycle in a gra[ph] with at most n vtxs

$|V(G)| = n$

② If $l > n$.
Long Simple path on graph G will always return false.

Construction of the input of LONGSIMPLECYCLE

Let G be the input graph for LongSimplePath, and let V be the vertex set with the size n. Let E be the edge set.

Construction of the input of LONGSIMPLECYCLE

Let G be the input graph for LongSimplePath, and let V be the vertex set with the size n. Let E be the edge set.

Construction of new graph G':

1. Copy all vertices and edges from G to G'

Construction of the input of LONGSIMPLECYCLE

Let G be the input graph for LongSimplePath, and let V be the vertex set with the size n. Let E be the edge set.

Construction of new graph G':

1. Copy all vertices and edges from G to G'

2. Add n more vertices to G' and create a simple directed path P: v' → u' by using these n vertices. W.L.O.G label the start vertex as v' and end vertex as u'. This path has length (n-1).

Construction of the input of LONGSIMPLECYCLE

Let G be the input graph for LongSimplePath, and let V be the vertex set with the size n. Let E be the edge set.

Construction of new graph G':

1. Copy all vertices and edges from G to G'

2. Add n more vertices to G' and create a simple directed path P: v' → u' by using these n vertices. W.L.O.G label the start vertex as v' and end vertex as u'. This path has length (n-1).

3. Connect G and P with two more edges: u' → u and v → v'.

Construction of the input of LONGSIMPLECYCLE

Let G be the input graph for LongSimplePath, and let V be the vertex set with the size n. Let E be the edge set.

Construction of new graph G':

1. Copy all vertices and edges from G to G'

2. Add n more vertices to G' and create a simple directed path P: v' → u' by using these n vertices. W.L.O.G label the start vertex as v' and end vertex as u'. This path has length (n-1).

3. Connect G and P with two more edges: u' → u and v → v'.

(This implies that there exists a simple path v→u in G' with length (n+1))

Construction of the input of LONGSIMPLECYCLE

Let G be the input graph for LongSimplePath, and let V be the vertex set with the size n. Let E be the edge set.

The number of vertice in cycle: $k + (n-1) + 2$

Running time of Construction of new input

Let G be the input graph for LongSimplePath, and let V be the vertex set with the size n. Let E be the edge set.

Construction of new graph G’ with G, P and 2 edges.

The number of vertice in cycle: $k + (n-1) + 2$

It takes $O(|V|+|E|)$

Analysis

Suppose $(G; u; v; k)$ is a YES-instance for LongSimplePath.

Analysis

Suppose (G; u; v; k) is a YES-instance for LongSimplePath.

So, there exists a simple path P1: u → v with length |P1|=k in graph G and G'. it contains (k+1) distinct vertices including u and v.

Analysis

Suppose (G; u; v; k) is a YES-instance for LongSimplePath.

So, there exists a simple path P1: u → v with length |P1|=k in graph G and G'. it contains (k+1) distinct vertices including u and v.

For G', there exists a simple path P2: v→u in G' with length |P2|=(n+1), passing through new added n vertices and u,v.

Analysis

Suppose (G; u; v; k) is a YES-instance for LongSimplePath.

So, there exists a simple path P1: u → v with length |P1|=k in graph G and G'. it contains (k+1) distinct vertices including u and v.

For G', there exists a simple path P2: v→u in G' with length |P2|=(n+1), passing through new added n vertices and u,v.

And we can get P1UP2 = {u,v}. Combine P1: u → v and P2: v→u together, we can get a simple cycle with length |P1|+|P2|=(n+k+1) and the total vertex number is (n+k+1) >= the input ℓ

Analysis

Suppose (G; u; v; k) is a YES-instance for LongSimplePath.

So, there exists a simple path P1: u → v with length |P1|=k in graph G and G'. it contains (k+1) distinct vertices including u and v.

For G', there exists a simple path P2: v→u in G' with length |P2|=(n+1), passing through new added n vertices and u,v.

And we can get P1UP2 = {u,v}. Combine P1: u → v and P2: v→u together, we can get a simple cycle with length |P1|+|P2|=(n+k+1) and the total vertex number is (n+k+1) >= the input ℓ

Therefore, it is a YES-instance for LongSimpleCycle.

Analysis

Suppose (G’; ℓ) is a YES-instance for LongSimpleCycle

Analysis

Suppose (G'; ℓ) is a YES-instance for LongSimpleCycle

So, there exists a simple cycle containing (n+k+1) vertices.

Analysis

Suppose (G'; ℓ) is a YES-instance for LongSimpleCycle

So, there exists a simple cycle containing (n+k+1) vertices.

Claim 1: This cycle must contain newly added vertices.

Analysis

Suppose (G’; ℓ) is a YES-instance for LongSimpleCycle

So, there exists a simple cycle containing (n+k+1) vertices.

Claim 1: This cycle must contain newly added vertices.

Prove: if not, there are at most n vertices in the remaining graphs, which cannot construct a (n+k+1)-simple cycle. Contradiciton.

Analysis

Suppose (G'; ℓ) is a YES-instance for LongSimpleCycle

So, there exists a simple cycle containing (n+k+1) vertices.

Claim 1: This cycle must contain newly added vertices.

Prove: if not, there are at most n vertices in the remaining graphs, which cannot construct a (n+k+1)-simple cycle. Contradiciton.

Claim 2: This cycle must contain all n newly added vertices and u, v.

Analysis

Suppose (G'; ℓ) is a YES-instance for LongSimpleCycle

So, there exists a simple cycle containing (n+k+1) vertices.

Claim 1: This cycle must contain newly added vertices.

Prove: if not, there are at most n vertices in the remaining graphs(G), which cannot construct a (n+k+1)-simple cycle. Contradiciton.

Claim 2: This cycle must contain all n newly added vertices and u, v.

Prove: since for each newly added vertex, it has only 2 edges adjacent to other new vertices or u or v. If any newly added vertex missing, it will not form a cycle.If u or v missing, it will not form a cycle.

Analysis

Suppose (G'; ℓ) is a YES-instance for LongSimpleCycle

So, there exists a simple cycle containing (n+k+1) vertices.

Claim 1: This cycle must contain newly added vertices.

Claim 2: This cycle must contain all n newly added vertices and u, v.

Analysis

Suppose (G'; ℓ) is a YES-instance for LongSimpleCycle

So, there exists a simple cycle containing (n+k+1) vertices.

Claim 1: This cycle must contain newly added vertices.

Claim 2: This cycle must contain all n newly added vertices and u, v.

Hence the simple cycle contains a simple path P1: v→ u with (n+2) vertices and length (n+1).

Analysis

Suppose (G'; ℓ) is a YES-instance for LongSimpleCycle

So, there exists a simple cycle containing (n+k+1) vertices.

Claim 1: This cycle must contain newly added vertices.

Claim 2: This cycle must contain all n newly added vertices and u, v.

Hence the simple cycle contains a simple path P1: v→ u with (n+2) vertices and length (n+1).

And the remaining graph is another simple path P2: u→v with length k without any newly-added vertices. Hence P2: u→v is on graph G.

Analysis

Suppose (G'; ℓ) is a YES-instance for LongSimpleCycle

So, there exists a simple cycle containing (n+k+1) vertices.

Claim 1: This cycle must contain newly added vertices.

Claim 2: This cycle must contain all n newly added vertices and u, v.

Hence the simple cycle contains a simple path P1: v→ u with (n+2) vertices and length (n+1).

And the remaining graph is another simple path P2: u→v with length k without any newly-added vertices. Hence P2: u→v is on graph G.

it is a YES-instance for LongSimplePath.

Q6 2)

Given that LongSimplePath is NP-hard. From part 1, show that there is no known polynomial time algorithm for LongSimpleCycle.

Proof by Contradiction.

If there was a known polynomial time algorithm for LongSimpleCycle, then you have a poly-nomial time algorithm for LongSimplePath:

Step 1: run the reduction in part 1 to convert (G; u; v; k) to (G'; ℓ).

Step 2: run the algorithm for LongSimpleCycle on (G'; ℓ).

This contradicts the fact that there are no polynomial time algorithms known for NP-hard problems.