Longest Path Problem

Ahnaf Faisal, 1505005
Raihanul Alam, 1505010
Mahim Mahbub, 1505022
Zahin Wahab, 1505031
Bishal Basak Papan, 1505043

Department of Computer Science and Engineering,
Bangladesh University of Engineering and Technology

September 21, 2020

Outline:

Project Management 101

- This is my first project as a manager.

Project Management 101

- This is my first project as a manager.
- Let's analyze the activities my team has to do for project completion.

Project Management 101

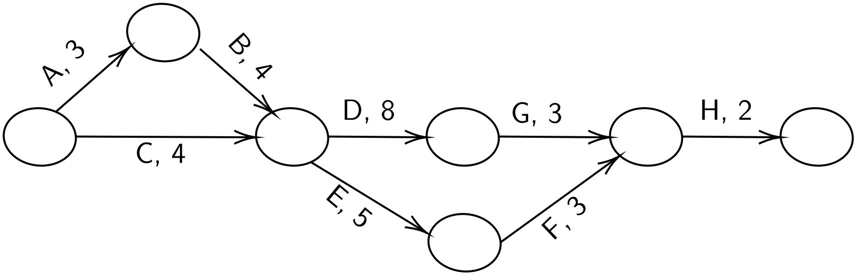
- This is my first project as a manager.
- Let's analyze the activities my team has to do for project completion.
- Wish me luck !

List of Activities

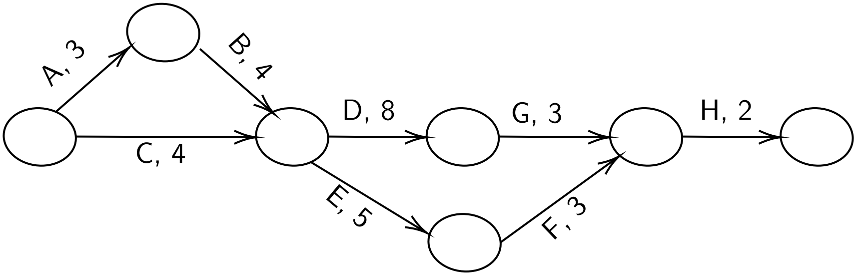
Activity ID	Activity	Duration	Dependencies
A	Market Survey	3	-
C	Provide questionnaires	4	-
B	Research company policies	4	A
D	Data flow analysis	8	B,C
E	Prototyping	5	B,C
F	Get feedback for prototype	3	E
G	Cost-benefit analysis	3	D
H	Prepare and present proposal	2	F,G

Table: Activities, their duration (weeks) and dependencies

Analysis using PERT Diagram

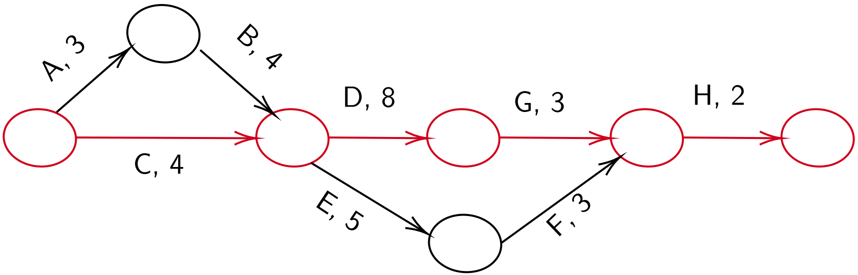


Analysis using PERT Diagram



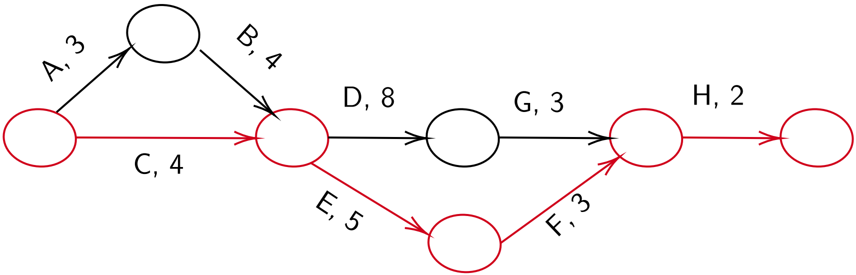
Time to spend resources to reduce total project time and get the well deserved bonus

Critical Path ?



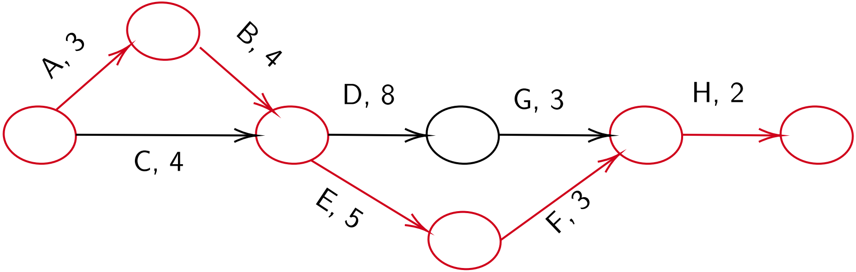
Path duration = 17 weeks

Critical Path ?



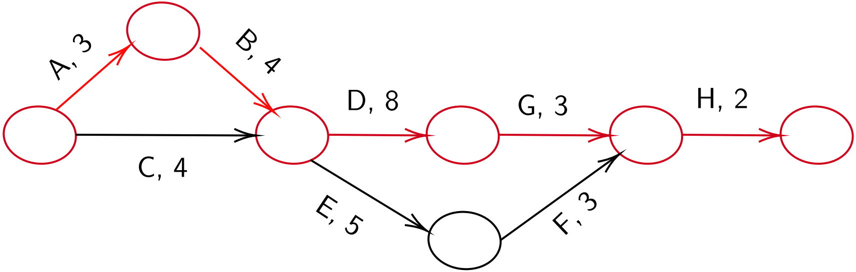
Path duration = 14 weeks

Critical Path ?



Path duration = 17 weeks

Critical Path ?



Path duration = 20 weeks !!

Longest Path Problem

- This directly leads to the problem we are tasked with analyzing: **Longest Path Problem**.

Longest Path Problem

This directly leads to the problem we are tasked with analyzing: **Longest Path Problem**.

Decision Version (Weighted)

Given a graph G, and an integer k, does the graph G have a longest path of *total weight* **at least** k ?

Longest Path Problem

This directly leads to the problem we are tasked with analyzing: **Longest Path Problem**.

Decision Version (Weighted)

Given a graph G, and an integer k, does the graph G have a longest path of *total weight* **at least** k ?

Decision Version (Un-Weighted)

Given a graph G, and an integer k, does the graph G have a longest path of *total number of edges* **at least** k ?

Longest Path Problem is NP-Complete

To prove longest path problem is NP-Complete, we need to prove that

Longest Path Problem is NP-Complete

To prove longest path problem is NP-Complete, we need to prove that

- it is in NP

Longest Path Problem is NP-Complete

To prove longest path problem is NP-Complete, we need to prove that

- it is in NP
- and it is NP-Hard.

Longest Path Problem is in NP

We now prove longest path problem ϵ NP

Longest Path Problem is in NP

We now prove longest path problem ϵ NP

- Given an integer k and a permutation of vertices $v_1, v_2....v_n$ for unweighted graph, we can calculate whether the path between v_1 and v_n is of length at least k.

Longest Path Problem is in NP

We now prove longest path problem ϵ NP

- Given an integer k and a permutation of vertices $v_1, v_2....v_n$ for unweighted graph, we can calculate whether the path between v_1 and v_n is of length at least k.
- Clearly this takes linear time.

Longest Path Problem is NP-Hard

We now show longest path problem is NP-Hard.

Longest Path Problem is NP-Hard

We now show longest path problem is NP-Hard.

- To do so, we show that the longest path problem is at least as hard as the 3-SAT problem.

Claim

3-SAT $\leq_p$ Longest Path

Reduction from 3-SAT formula

- At first, we need to convert an instance of $3-SAT$ problem to an instance of $LongestPath$ problem.

Reduction from 3-SAT formula

- At first, we need to convert an instance of $3 - SAT$ problem to an instance of $LongestPath$ problem.
- As the instance of $LongestPath$ problem must be a graph, we will show the process of converting a $3 - SAT$ equation, Y to a $LongestPath$ problem instance, which will be a graph $G = (V, E)$.

Reduction from 3-SAT formula

- At first, we need to convert an instance of $3 - SAT$ problem to an instance of $LongestPath$ problem.
- As the instance of $LongestPath$ problem must be a graph, we will show the process of converting a $3 - SAT$ equation, Y to a $LongestPath$ problem instance, which will be a graph $G = (V, E)$.
- This construction takes polynomial time.

Reduction from 3-SAT formula

- At first, we need to convert an instance of $3-SAT$ problem to an instance of $LongestPath$ problem.
- As the instance of $LongestPath$ problem must be a graph, we will show the process of converting a $3-SAT$ equation, Y to a $LongestPath$ problem instance, which will be a graph $G=(V,E)$.
- This construction takes polynomial time.

Value of k

$k = 2 + 2n + (3+n)q$

Claim

Y is satisfiable if and only if G has a path of length at least k.

Reduction from 3-SAT formula

- At first, we need to convert an instance of $3-SAT$ problem to an instance of $LongestPath$ problem.
- As the instance of $LongestPath$ problem must be a graph, we will show the process of converting a $3-SAT$ equation, Y to a $LongestPath$ problem instance, which will be a graph $G=(V,E)$.
- This construction takes polynomial time.

Value of k

$k=2+2n+(3+n)q$

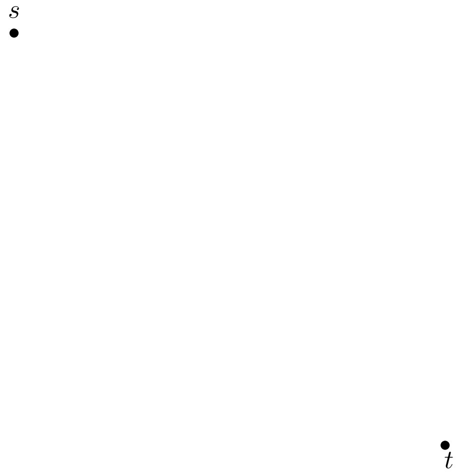
Claim

Y is satisfiable if and only if G has a path of length at least k.

- Now we will show the conversion steps from $3-SAT$ to $LongestPath$.

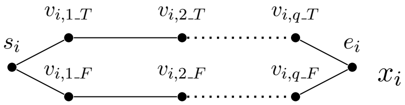
Graph Construction

- There is a source vertex s and a destination vertex t in G.



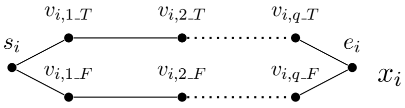
Graph Construction

- For each variable x_i in Y, where $1 \leq i \leq n$, we construct a loop like this

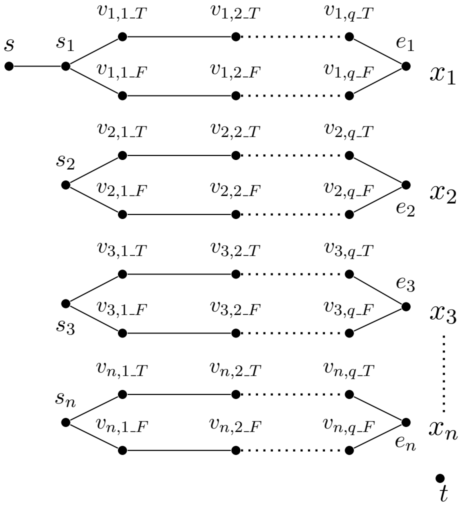


Graph Construction

- Every loop has a start vertex, s_i, and an end vertex, e_i.
- Each loop contains two paths: upper and lower.
- Each vertex in the path corresponds to the {0,1} or {True, False} assignment of the variable in a clause c_j, where $1 \leq j \leq q$.
- Number of vertices in each path is $q + 2$.

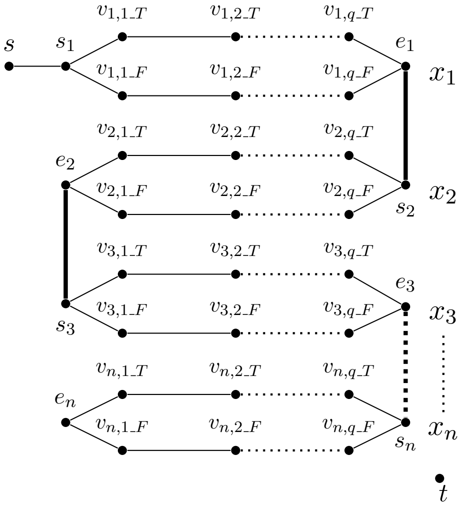


Graph Construction



- n number of loops between s and t.
- Start node of the first variable will be adjacent to s.

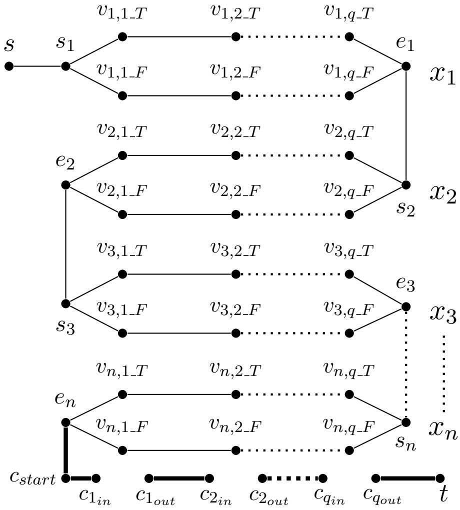
Graph Construction



- Edge is added between the end vertex of a variable and the start vertex of its next variable.

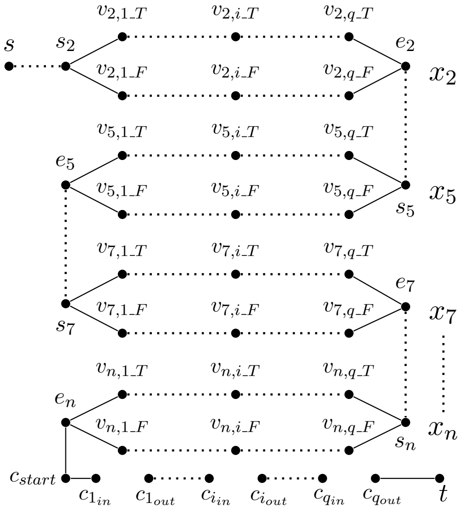
Graph Construction

- Two vertices (c_{in} and c_{out}) are added per clause, c_j where $1 \leq j \leq q$.
- Edges are added between the out vertex of one clause to the in vertex of next clause.



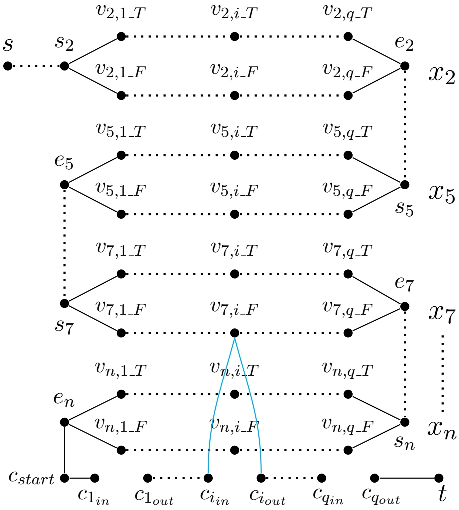
Graph Construction

- We will add a *gadget* for each clause in Y.
- For example, if the i^{th} clause had the form $x_7 \bigvee \overline{x_5} \bigvee x_2$ then



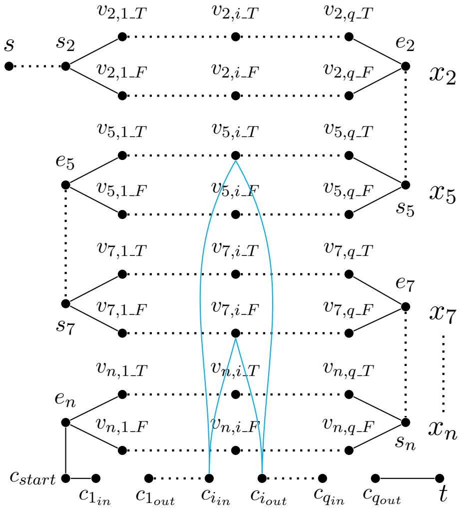
Graph Construction

- We will add a *gadget* for each clause in Y.
- For example, if the i^{th} clause had the form $x_7 \bigvee \overline{x_5} \bigvee x_2$ then
- one path would go through a vertex in the lower loop corresponding to x_7,



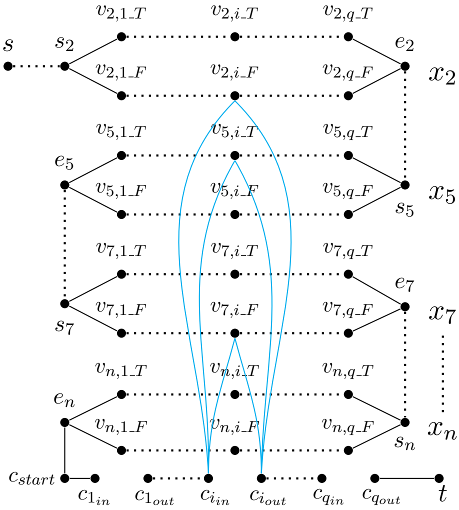
Graph Construction

- We will add a *gadget* for each clause in Y.
- For example, if the i^{th} clause had the form $x_7 \bigvee \overline{x_5} \bigvee x_2$ then
  - one path would go through a vertex in the lower loop corresponding to x_7,
  - another path would go through a vertex in the upper loop corresponding to x_5 and



Graph Construction

- We will add a *gadget* for each clause in Y.
- For example, if the i^{th} clause had the form $x_7 \bigvee \overline{x_5} \bigvee x_2$ then
  - one path would go through a vertex in the lower loop corresponding to x_7,
  - another path would go through a vertex in the upper loop corresponding to x_5 and
  - the third would go through the lower loop corresponding to x_2.



Graph Construction

- Let the 3-SAT equation be,
  $Y = \left(x_1 \bigvee \overline{x_4} \bigvee x_3\right) \bigwedge$
  $\left(\overline{x_1} \bigvee x_2 \bigvee \overline{x_3}\right) \bigwedge$
  $\left(x_2 \bigvee x_3 \bigvee \overline{x_4}\right)$

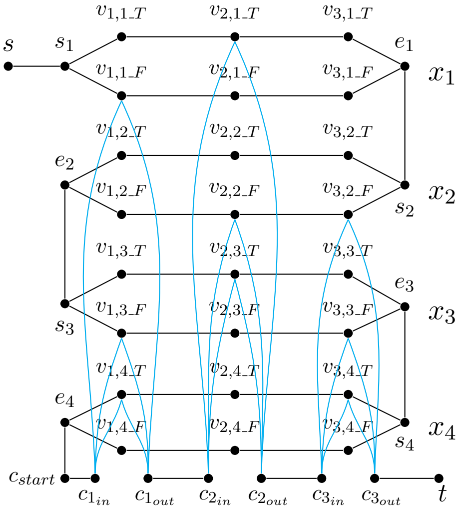
Graph Construction

- Let the 3-SAT equation be,
  $Y = \left(x_1 \bigvee \overline{x_4} \bigvee x_3\right) \bigwedge$
  $\left(\overline{x_1} \bigvee x_2 \bigvee \overline{x_3}\right) \bigwedge$
  $\left(x_2 \bigvee x_3 \bigvee \overline{x_4}\right)$

- $c_1 = \left(x_1 \bigvee \overline{x_4} \bigvee x_3\right)$
- $c_2 = \left(\overline{x_1} \bigvee x_2 \bigvee \overline{x_3}\right)$
- $c_3 = \left(x_2 \bigvee x_3 \bigvee \overline{x_4}\right)$

Graph Construction

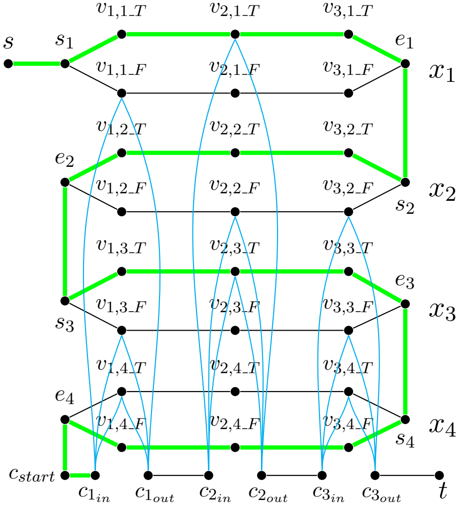
- Let the 3-SAT equation be, $Y = (x_1 \bigvee \overline{x_4} \bigvee x_3) \bigwedge (\overline{x_1} \bigvee x_2 \bigvee \overline{x_3}) \bigwedge (x_2 \bigvee x_3 \bigvee \overline{x_4})$
  - $c_1 = (x_1 \bigvee \overline{x_4} \bigvee x_3)$
  - $c_2 = (\overline{x_1} \bigvee x_2 \bigvee \overline{x_3})$
  - $c_3 = (x_2 \bigvee x_3 \bigvee \overline{x_4})$
- Then the corresponding graph G according to the $LongestPath$ formulation of Y will be:



Y: $(x_1 \vee \overline{x_4} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_3 \vee \overline{x_4})$

Finding the Path

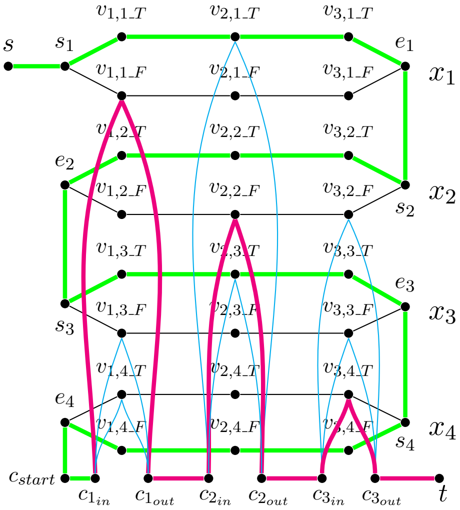
- Starting from s, we take exactly one path, either True Path or False Path from each loop representing a variable x_i. We color this path with $green$.
- According to this way, after reaching the end vertex of the n^{th} variable, e_n, we add c_{start} and $c_{1_{in}}$ to the path.



Y: $(x_1 \vee \overline{x_4} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_3 \vee \overline{x_4})$

Finding the Path

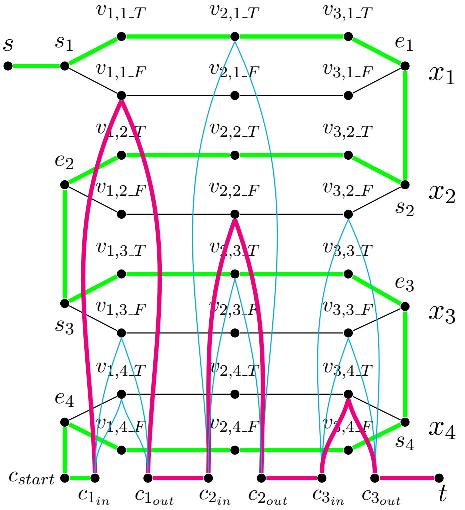
- For the $gadget$ corresponding to each clause C_j, there are three choices.
- For each clause, we take any of the three choices so that no node on the $green$ path is again traversed by the chosen path from the $gadget$. We color this path with $pink$.
- We add t to this path.



Y: $(x_1 \vee \overline{x_4} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_3 \vee \overline{x_4})$

Finding the Path

- We denote the path obtained in this way as P.
- Let the length of P be k.



Y: $(x_1 \vee \overline{x_4} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_3 \vee \overline{x_4})$

Reduction from 3-SAT formula

Clearly this construction is polynomial.

Set k

$k = n(q+1) + n - 1 + 2q + q - 1 + 4$

Reduction from 3-SAT formula

Clearly this construction is polynomial.

Set k

$k = n(q+1) + n - 1 + 2q + q - 1 + 4$

Simplifying

$k = 2 + 2n + (3+n)q$

Reduction from 3-SAT formula

Clearly this construction is polynomial.

Set k

$k = n(q+1) + n - 1 + 2q + q - 1 + 4$

Simplifying

$k = 2 + 2n + (3+n)q$

Claim

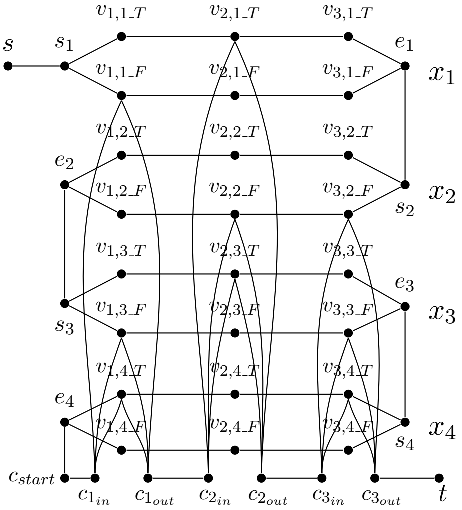
Y is satisfiable if and only if G has a path of length of at least k.

Reduction from 3-SAT formula

Necessity

If Y is satisfiable, then G has a path of length of at least k.

- Since Y is satisfiable, each clause c_j (where $1 \leq j \leq q$) is true. There must be at least one true literal per clause.



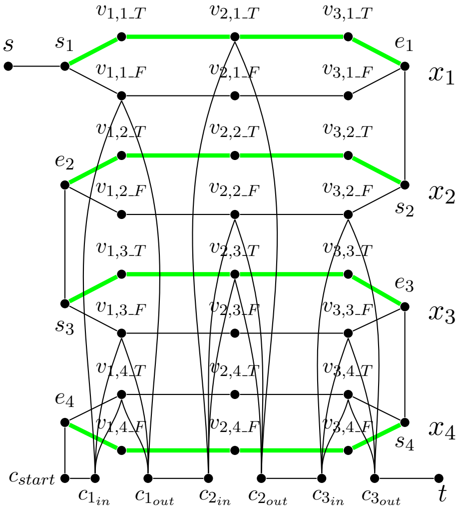
Y: $(x_1 \vee \overline{x_4} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_3 \vee \overline{x_4})$

Reduction from 3-SAT formula

Necessity

If Y is satisfiable, then G has a path of length of at least k.

- Since Y is satisfiable, every variable must have an assignment ($True$ or $False$). So upper path of variable x_i is taken if it is true and lower path is taken if it is false. Thus for every variable, either lower or upper path is taken.



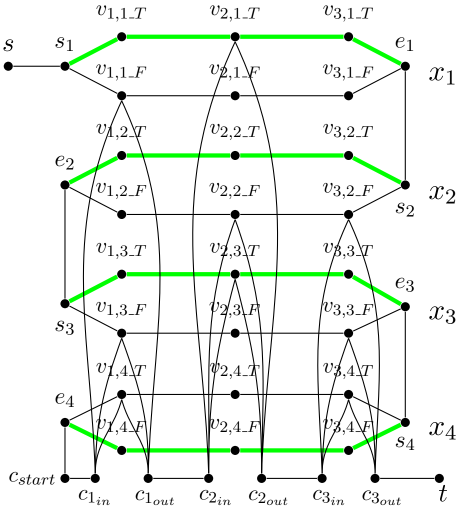
Y: $(x_1 \vee \overline{x_4} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_3 \vee \overline{x_4})$

Reduction from 3-SAT formula

Necessity

If Y is satisfiable, then G has a path of length of at least k.

- Length of a path between the start and the end vertex in one loop is $q+1$. Thus total $n*(q+1)$ edges are taken.
- To construct a path, we need to take edges from ending vertex of each loop to starting vertex of next loop.
- So total $n-1$ edges are taken.



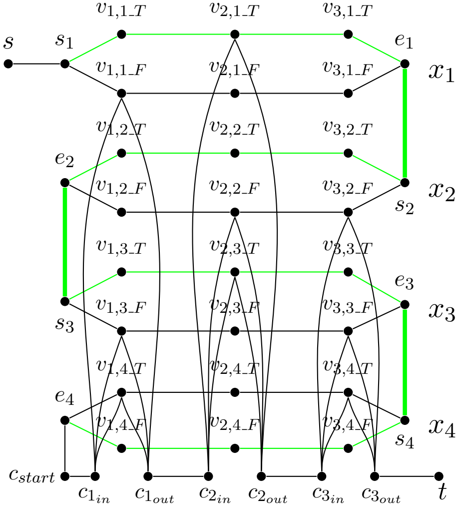
Y: $(x_1 \vee \overline{x_4} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_3 \vee \overline{x_4})$

Reduction from 3-SAT formula

Necessity

If Y is satisfiable, then G has a path of length of at least k.

- Length of a path between the start and the end vertex in one loop is $q+1$. Thus total $n*(q+1)$ edges are taken.
- To construct a path, we need to take edges from ending vertex of each loop to starting vertex of next loop.
- So total $n-1$ edges are taken.



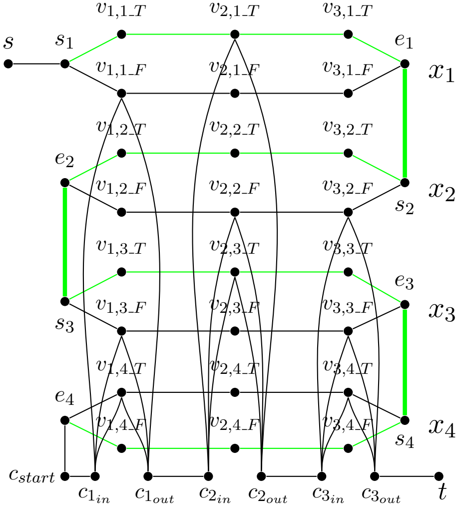
Y: $(x_1 \vee \overline{x_4} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_3 \vee \overline{x_4})$

Reduction from 3-SAT formula

Necessity

If Y is satisfiable, then G has a path of length of at least k.

- Length of a path between the start and the end vertex in one loop is $q+1$. Thus total $n*(q+1)$ edges are taken.
- To construct a path, we need to take edges from ending vertex of each loop to starting vertex of next loop.
- So total $n-1$ edges are taken.



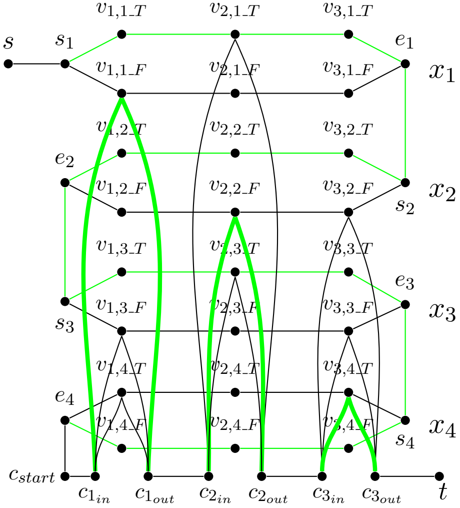
Y: $(x_1 \vee \overline{x_4} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_3 \vee \overline{x_4})$

Reduction from 3-SAT formula

Necessity

If Y is satisfiable, then G has a path of length of at least k.

- If the term x_i is true and clause c_j has the term x_i, we take edges from $c_{j_{in}}$ to x_{i,j_F} and from x_{i,j_F} to $c_{j_{out}}$. And if the term x_i is false and clause c_j has the term $\overline{x_i}$, we need to take edges from $c_{j_{in}}$ to x_{i,j_T} and from x_{i,j_T} to $c_{j_{out}}$.
- Thus for q clauses, total $2 * q$ edges are taken.



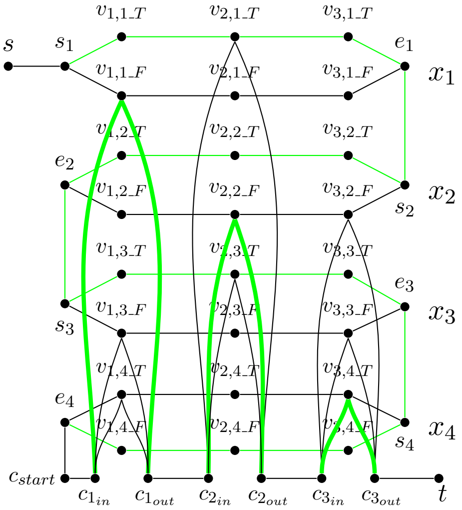
Y: $(x_1 \vee \overline{x_4} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_3 \vee \overline{x_4})$

Reduction from 3-SAT formula

Necessity

If Y is satisfiable, then G has a path of length of at least k.

- If the term x_i is true and clause c_j has the term x_i, we take edges from $c_{j_{in}}$ to x_{i,j_F} and from x_{i,j_F} to $c_{j_{out}}$. And if the term x_i is false and clause c_j has the term $\overline{x_i}$, we need to take edges from $c_{j_{in}}$ to x_{i,j_T} and from x_{i,j_T} to $c_{j_{out}}$.
- Thus for q clauses, total $2 * q$ edges are taken.



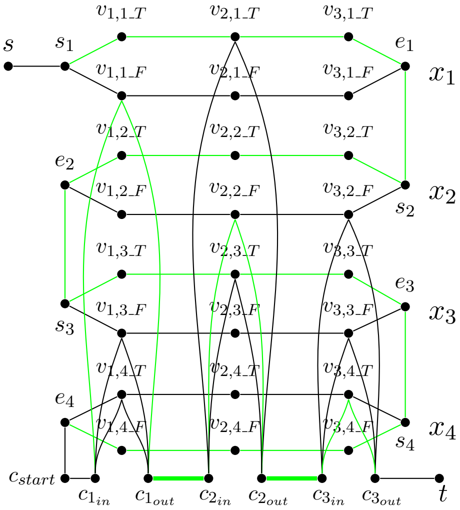
Y: $(x_1 \vee \overline{x_4} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_3 \vee \overline{x_4})$

Reduction from 3-SAT formula

Necessity

If Y is satisfiable, then G has a path of length of at least k.

- To complete the path, edges from $c_{j_{out}}$ to $c_{j+1_{in}}$ must be taken.
- Total $q-1$ edges are taken.
- Edges from s to s_1, e_n to c_{start}, c_{start} to $c_{1_{in}}$ and $c_{q_{out}}$ to t is taken.
- Thus total $n*(q+1)+n-1+2*q+4$ edges is taken. So the length of the path is k.



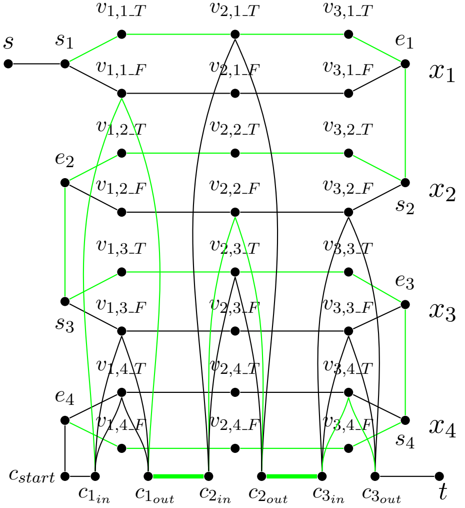
Y: $(x_1 \vee \overline{x_4} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_3 \vee \overline{x_4})$

Reduction from 3-SAT formula

Necessity

If Y is satisfiable, then G has a path of length of at least k.

- To complete the path, edges from $c_{j_{out}}$ to $c_{j+1_{in}}$ must be taken.
- Total $q-1$ edges are taken.
- Edges from s to s_1, e_n to c_{start}, c_{start} to $c_{1_{in}}$ and $c_{q_{out}}$ to t is taken.
- Thus total $n*(q+1)+n-1+2*q+4$ edges is taken. So the length of the path is k.



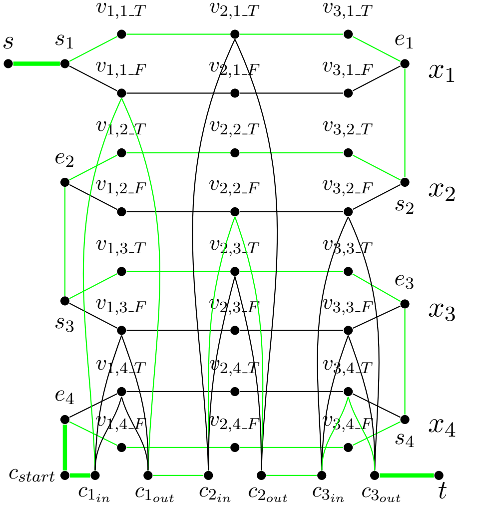
Y: $(x_1 \vee \overline{x_4} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_3 \vee \overline{x_4})$

Reduction from 3-SAT formula

Necessity

If Y is satisfiable, then G has a path of length of at least k.

- To complete the path, edges from $c_{j_{out}}$ to $c_{j+1_{in}}$ must be taken.
- Total $q-1$ edges are taken.
- Edges from s to s_1, e_n to c_{start}, c_{start} to $c_{1_{in}}$ and $c_{q_{out}}$ to t is taken.
- Thus total $n*(q+1)+n-1+2*q+4$ edges is taken. So the length of the path is k.



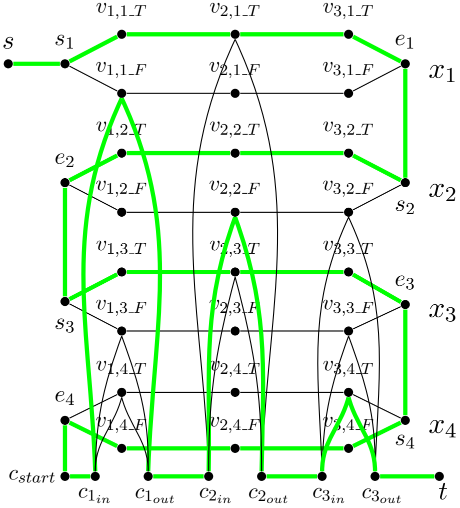
Y: $(x_1 \vee \overline{x_4} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_3 \vee \overline{x_4})$

Reduction from 3-SAT formula

Necessity

If Y is satisfiable, then G has a path of length of at least k.

- To complete the path, edges from $c_{j_{out}}$ to $c_{j+1_{in}}$ must be taken.
- Total $q-1$ edges are taken.
- Edges from s to s_1, e_n to c_{start}, c_{start} to $c_{1_{in}}$ and $c_{q_{out}}$ to t is taken.
- Thus total $n*(q+1)+n-1+2*q+4$ edges is taken. So the length of the path is k.



Y: $(x_1 \vee \overline{x_4} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_3 \vee \overline{x_4})$

Reduction from 3-SAT formula

Sufficiency

If G has a path of length at least k, then Y is satisfiable.

- Since G has a path P of length at least k, the value assigned to each literal x_i will have a corresponding path($True$ or $False$) to it.
- For a literal, x_i, we assign $True$ to x_i if P contains the upper/true path in the loop of x_i and assign $False$ if P contains the lower path.
- Due to the construction of G, for each literal in clause c_j, the complement term will have two edges with that clause, one edge with $c_{j_{in}}$ and another edge with $c_{j_{out}}$.
- If a clause c_j contains the literal x_i, then $True$ is assigned to x_i which will make c_j $True$. And if a clause c_j contains the literal $\overline{x_i}$, then the value assigned to $\overline{x_i}$ is $False$ which will lead c_j to being $True$.
- This will hold for each clause in Y. So Y is satisfiable.

Reduction from 3-SAT formula

Sufficiency

If G has a path of length at least k, then Y is satisfiable.

- Since G has a path P of length at least k, the value assigned to each literal x_i will have a corresponding path($True$ or $False$) to it.
- For a literal, x_i, we assign $True$ to x_i if P contains the upper/true path in the loop of x_i and assign $False$ if P contains the lower path.
- Due to the construction of G, for each literal in clause c_j, the complement term will have two edges with that clause, one edge with $c_{j_{in}}$ and another edge with $c_{j_{out}}$.
- If a clause c_j contains the literal x_i, then $True$ is assigned to x_i which will make c_j $True$. And if a clause c_j contains the literal $\overline{x_i}$, then the value assigned to $\overline{x_i}$ is $False$ which will lead c_j to being $True$.
- This will hold for each clause in Y. So Y is satisfiable.

Reduction from 3-SAT formula

Sufficiency

If G has a path of length at least k, then Y is satisfiable.

- Since G has a path P of length at least k, the value assigned to each literal x_i will have a corresponding path($True$ or $False$) to it.
- For a literal, x_i, we assign $True$ to x_i if P contains the upper/true path in the loop of x_i and assign $False$ if P contains the lower path.
- Due to the construction of G, for each literal in clause c_j, the complement term will have two edges with that clause, one edge with $c_{j_{in}}$ and another edge with $c_{j_{out}}$.
- If a clause c_j contains the literal x_i, then $True$ is assigned to x_i which will make c_j $True$. And if a clause c_j contains the literal $\overline{x_i}$, then the value assigned to $\overline{x_i}$ is $False$ which will lead c_j to being $True$.
- This will hold for each clause in Y. So Y is satisfiable.

Reduction from 3-SAT formula

Sufficiency

If G has a path of length at least k, then Y is satisfiable.

- Since G has a path P of length at least k, the value assigned to each literal x_i will have a corresponding path($True$ or $False$) to it.
- For a literal, x_i, we assign $True$ to x_i if P contains the upper/true path in the loop of x_i and assign $False$ if P contains the lower path.
- Due to the construction of G, for each literal in clause c_j, the complement term will have two edges with that clause, one edge with $c_{j_{in}}$ and another edge with $c_{j_{out}}$.
- If a clause c_j contains the literal x_i, then $True$ is assigned to x_i which will make c_j $True$. And if a clause c_j contains the literal $\overline{x_i}$, then the value assigned to $\overline{x_i}$ is $False$ which will lead c_j to being $True$.
- This will hold for each clause in Y. So Y is satisfiable.

Reduction from 3-SAT formula

Sufficiency

If G has a path of length at least k, then Y is satisfiable.

- Since G has a path P of length at least k, the value assigned to each literal x_i will have a corresponding path($True$ or $False$) to it.
- For a literal, x_i, we assign $True$ to x_i if P contains the upper/true path in the loop of x_i and assign $False$ if P contains the lower path.
- Due to the construction of G, for each literal in clause c_j, the complement term will have two edges with that clause, one edge with $c_{j_{in}}$ and another edge with $c_{j_{out}}$.
- If a clause c_j contains the literal x_i, then $True$ is assigned to x_i which will make c_j $True$. And if a clause c_j contains the literal $\overline{x_i}$, then the value assigned to $\overline{x_i}$ is $False$ which will lead c_j to being $True$.
- This will hold for each clause in Y. So Y is satisfiable.

Polynomial time variations

Special graphs	Complexity	Comments
Tree	Linear	Dijkstra's algorithm [1]
Cacti Graph	$O(n^2)$	[2]
Bipartite Permutation Graph	Linear	[3]
Directed Acyclic Graph	Linear	Dynamic approach
Interval Graph	$O(n^4)$	Dynamic approach [4]
Circular Arc Graph	$O(n^4)$	Dynamic approach [5]
Co-compatibility Graph	$O(n^7)$	From Hasse diagram [6]

- Tree is undirected acyclic graph.Dijkstra proposed an algorithm for this in 1960.

Polynomial time variations

Special graphs	Complexity	Comments
Tree	Linear	Dijkstra's algorithm [1]
Cacti Graph	$O(n^2)$	[2]
Bipartite Permutation Graph	Linear	[3]
Directed Acyclic Graph	Linear	Dynamic approach
Interval Graph	$O(n^4)$	Dynamic approach [4]
Circular Arc Graph	$O(n^4)$	Dynamic approach [5]
Co-compatibility Graph	$O(n^7)$	From Hasse diagram [6]

- Cacti is a special kind of block graph which each block is a cycle.Two cycle share at most one vertex which is a separator.

Polynomial time variations

Special graphs	Complexity	Comments
Tree	Linear	Dijkstra's algorithm [1]
Cacti Graph	$O(n^2)$	[2]
Bipartite Permutation Graph	Linear	[3]
Directed Acyclic Graph	Linear	Dynamic approach
Interval Graph	$O(n^4)$	Dynamic approach [4]
Circular Arc Graph	$O(n^4)$	Dynamic approach [5]
Co-compatibility Graph	$O(n^7)$	From Hasse diagram [6]

- The class of bipartite permutation graphs is the intersection of two well known graph classes: bipartite graphs and permutation graphs. A complete bipartite decomposition of a bipartite permutation graph is proposed in this reference.

Polynomial time variations

Special graphs	Complexity	Comments
Tree	Linear	Dijkstra's algorithm [1]
Cacti Graph	$O(n^2)$	[2]
Bipartite Permutation Graph	Linear	[3]
Directed Acyclic Graph	Linear	Dynamic approach
Interval Graph	$O(n^4)$	Dynamic approach [4]
Circular Arc Graph	$O(n^4)$	Dynamic approach [5]
Co-compatibility Graph	$O(n^7)$	From Hasse diagram [6]

- The longest path for general graphs does not have a optimal substructure property but it has for weighted directed acyclic graphs.

Polynomial time variations

Special graphs	Complexity	Comments
Tree	Linear	Dijkstra's algorithm [1]
Cacti Graph	$O(n^2)$	[2]
Bipartite Permutation Graph	Linear	[3]
Directed Acyclic Graph	Linear	Dynamic approach
Interval Graph	$O(n^4)$	Dynamic approach [4]
Circular Arc Graph	$O(n^4)$	Dynamic approach [5]
Co-compatibility Graph	$O(n^7)$	From Hasse diagram [6]

- An interval graph is an undirected graph formed from a set of intervals on the real line, with a vertex for each interval and an edge between vertices whose intervals intersect. It is the intersection graph of the intervals.

Polynomial time variations

Special graphs	Complexity	Comments
Tree	Linear	Dijkstra's algorithm [1]
Cacti Graph	$O(n^2)$	[2]
Bipartite Permutation Graph	Linear	[3]
Directed Acyclic Graph	Linear	Dynamic approach
Interval Graph	$O(n^4)$	Dynamic approach [4]
Circular Arc Graph	$O(n^4)$	Dynamic approach [5]
Co-compatibility Graph	$O(n^7)$	From Hasse diagram [6]

- In graph theory, a circular-arc graph is the intersection graph of a set of arcs on the circle.It has one vertex for each arc in the set, and an edge between every pair of vertices corresponding to arcs that intersect.

Polynomial time variations

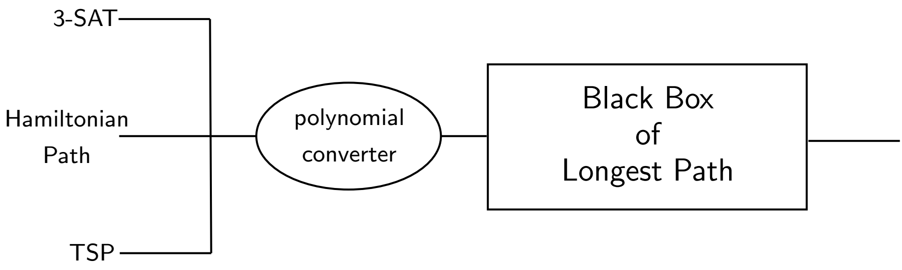
Special graphs	Complexity	Comments
Tree	Linear	Dijkstra's algorithm [1]
Cacti Graph	$O(n^2)$	[2]
Bipartite Permutation Graph	Linear	[3]
Directed Acyclic Graph	Linear	Dynamic approach
Interval Graph	$O(n^4)$	Dynamic approach [4]
Circular Arc Graph	$O(n^4)$	Dynamic approach [5]
Co-compatibility Graph	$O(n^7)$	From Hasse diagram [6]

- In graph theory, a comparability graph is an undirected graph that connects pairs of elements that are comparable to each other in a partial order.

Observation

- The longest path problem is solvable in polynomial time on any class of graphs with bounded tree-width or bounded clique-width, such as the distance-hereditary graphs.
- Finally, it is clearly NP-hard on all graph classes on which the Hamiltonian path problem is NP-hard, such as on split graphs, circle graphs, and planar graphs.

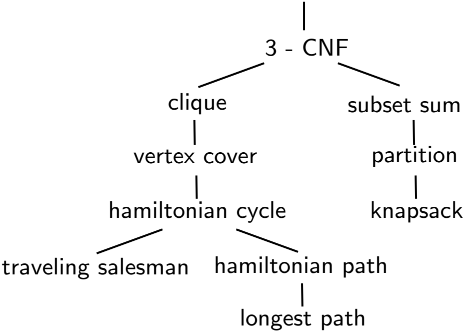
Problems that Reduce To Longest Path Problem



- The following diagram shows that 3-SAT, Hamiltonian Path, Traveling Salesman Problem are polynomial time reducible to Longest Path Problem.

Problems that Reduced To Longest Path Problem

Mother Problem:



- We could not find any problem which is reduced to longest path problem. Because, as the tree moves down, the problems tend to get harder from the previous ones.
- As longest path problem is harder than any known NP-hard problems, we couldn't find any problem that reduced from longest path problem.

References I

R.W. Bulterman, F.W. van der Sommen, G. Zwaan, T. Verhoeff, A.J.M. van Gasteren, and W.H.J. Feijen.
On computing a longest path in a tree.
Information Processing Letters, 81(2):93 – 96, 2002.

Ryuhei Uehara and Yushi Uno.
Efficient algorithms for the longest path problem.
In Rudolf Fleischer and Gerhard Trippen, editors, *Algorithms and Computation*, pages 871–883, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

Ryuhei Uehara and Gabriel Valiente.
Linear structure of bipartite permutation graphs and the longest path problem.
Information Processing Letters, 103(2):71 – 77, 2007.

References II

Kyriaki Ioannidou, George Mertzios, and Stavros Nikolopoulos.
The longest path problem is polynomial on interval graphs.
pages 403–414, 08 2009.

George B. Mertzios and Ivona Bezáková.
Computing and counting longest paths on circular-arc graphs in polynomial time.
Discrete Applied Mathematics, 164:383 – 399, 2014.
LAGOS'11: Sixth Latin American Algorithms, Graphs, and Optimization Symposium, Bariloche, Argentina — 2011.

Kyriaki Ioannidou and Stavros D. Nikolopoulos.
The longest path problem is polynomial on cocomparability graphs.
Algorithmica, 65(1):177–205, Jan 2013.