CS5371
Theory of Computation

Lecture 22: Complexity VII
(More NP-complete Problems)

Objectives

- We shall continue to look at more NP-complete problems:
  - Directed Hamiltonian Path
  - Hamiltonian Path
  - SUBSET SUM
  - PARTITION
- Some more if we have time today

Directed HAMPATH

Let G be a directed graph. A directed Hamiltonian path in G is a path that visits all the vertices of G once and only once.

Let D-HAMPATH be the language

$\{ \langle G,s,t \rangle \mid G$ has a directed Hamiltonian path from s to $t \}$

Theorem: D-HAMPATH is NP-complete.

D-HAMPATH is NP-complete (2)

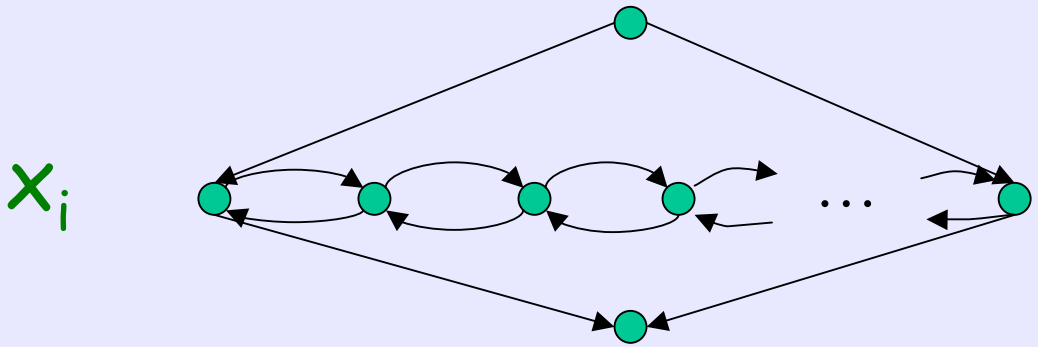
Proof: First, D-HAMPATH is in NP (easy to show). Then, we show it is NP-complete by reduction from 3SAT.

To determine if $\langle F \rangle$ is in 3SAT, we shall construct G with two special vertices s, t such that

$$\langle F \rangle \in \text{3SAT} \Leftrightarrow \langle G, s, t \rangle \in \text{D-HAMPATH}$$

D-HAMPATH is NP-complete (3)

Let $x_1, x_2, \ldots, x_j$ be the variables in the 3cnf-formula F. For each variable x_i, we create a 'diamond-shaped structure' that contains a horizontal row of nodes:



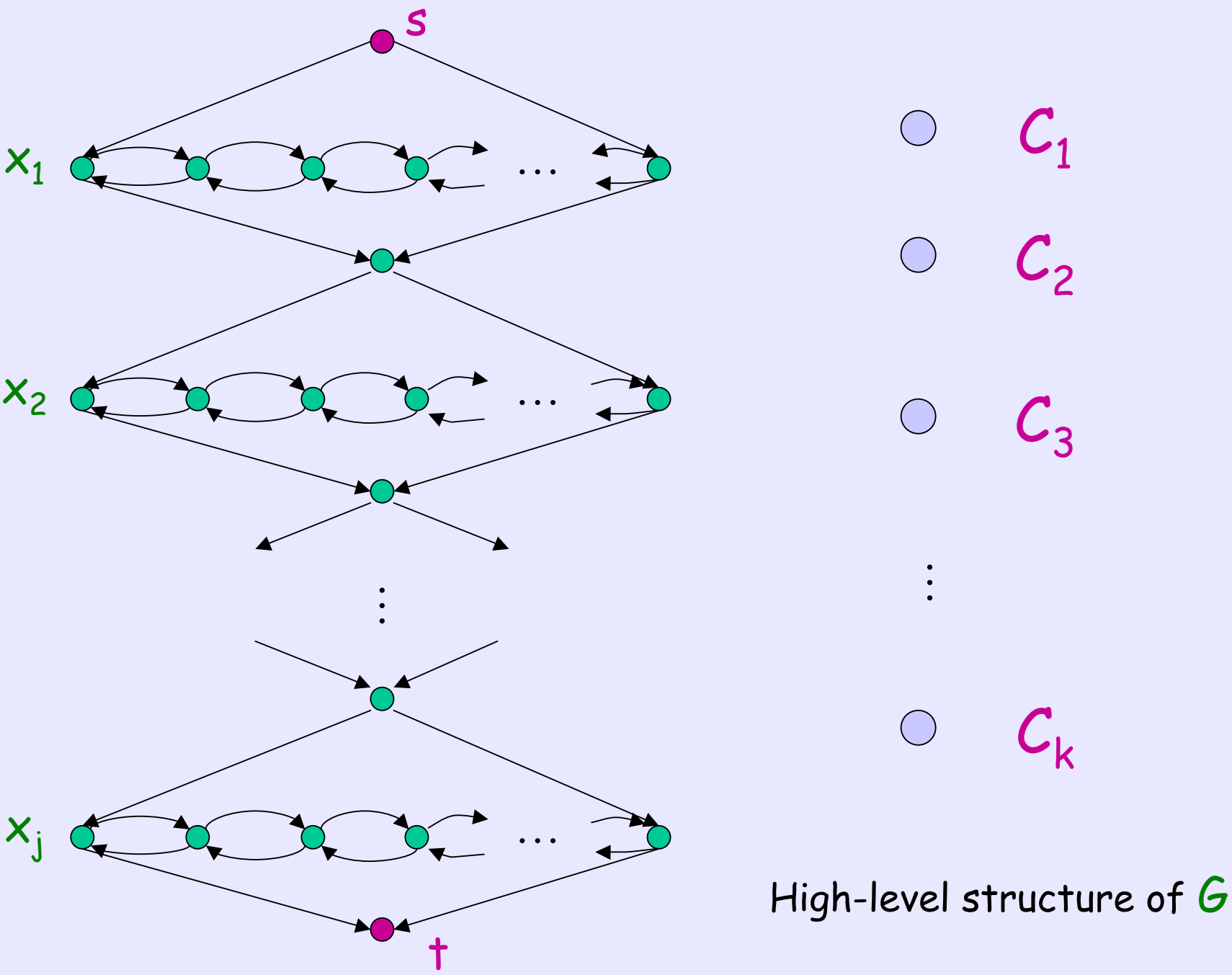
D-HAMPATH is NP-complete (4)

Let $C_1, C_2, \ldots, C_k$ be the clauses in F. For each clause C_m, we create a node:

C_m

Next slide shows the global structure of G.

It shows all the elements of G and their relationship, except the relationship of the variables to the clauses that contains them



High-level structure of G

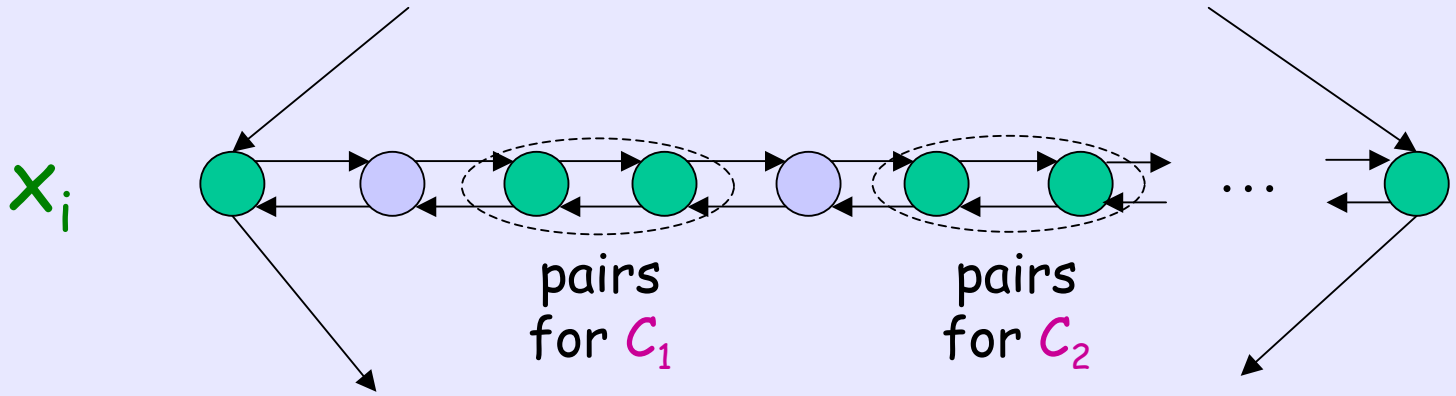
Diamond Structure

Each diamond structure contains a horizontal row of nodes connected by edges running in both directions.

There are $3k+1$ nodes in each row (in addition to the two nodes on the end that belongs to the diamond)

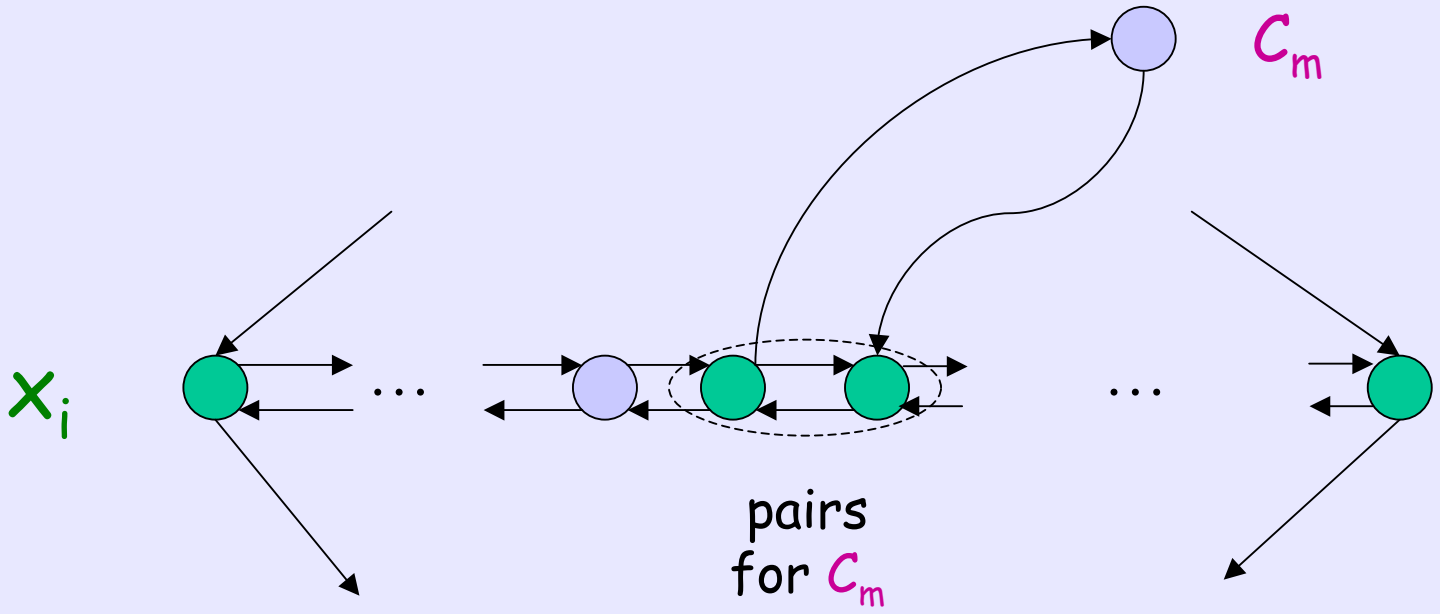
Diamond Structure

The nodes in the horizontal row are grouped into adjacent pairs, one for each clause, with extra separator nodes next to the pair:



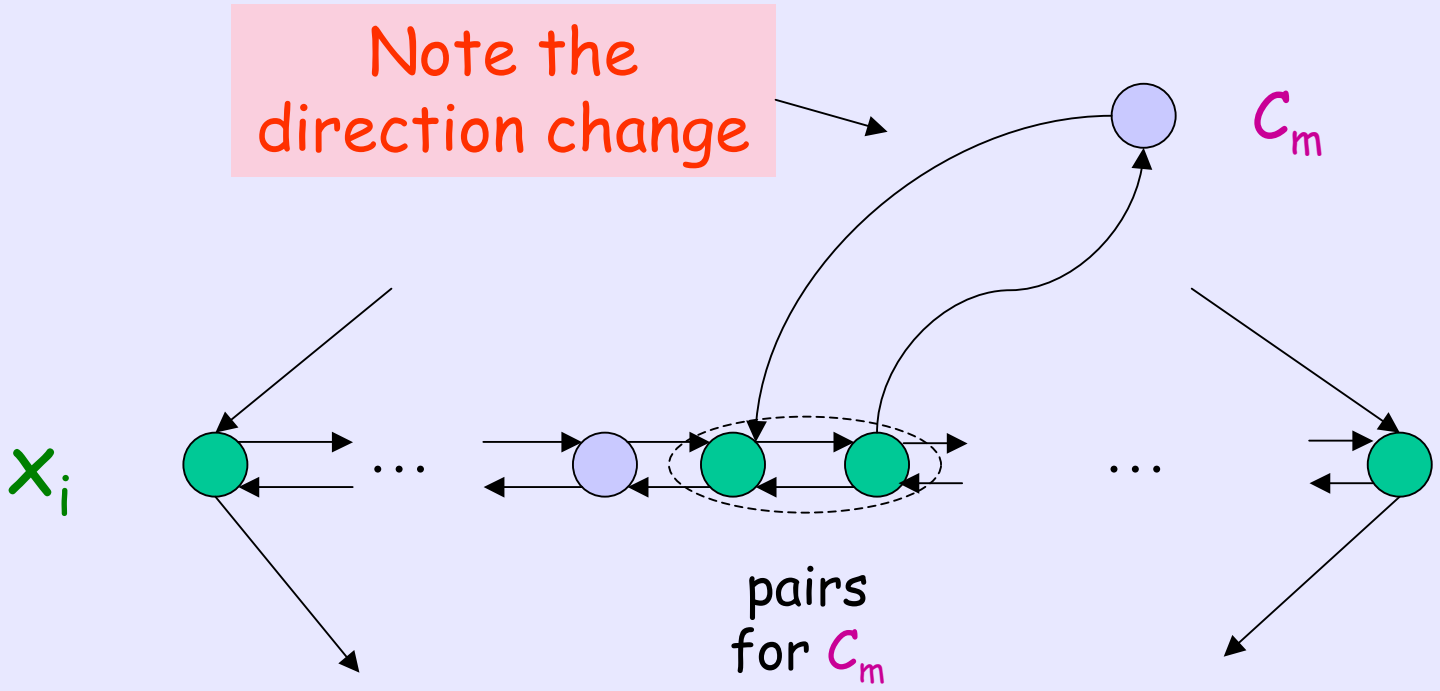
Connection with Clauses

If variable x_i appears in clause C_m, we add the following two edges from the m^{th} pair in the x_i's diamond structure to node C_m



Connection with Clauses (2)

If $\neg x_i$ appears in clause C_m, we add the following two edges from the m^{th} pair in the x_i's diamond structure to node C_m



D-HAMPATH is NP-complete (5)

After we add all the edges corresponding to each occurrence of x_i or $\neg x_i$ in each clause, the construction of G is finished

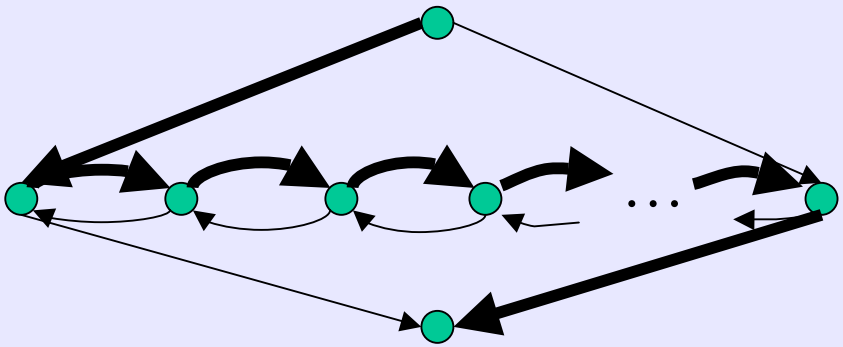
We claim that this is our desired reduction:

(=>) Suppose that F is satisfiable. To demonstrate an Hamiltonian path from s to t in G, we first ignore the clause nodes

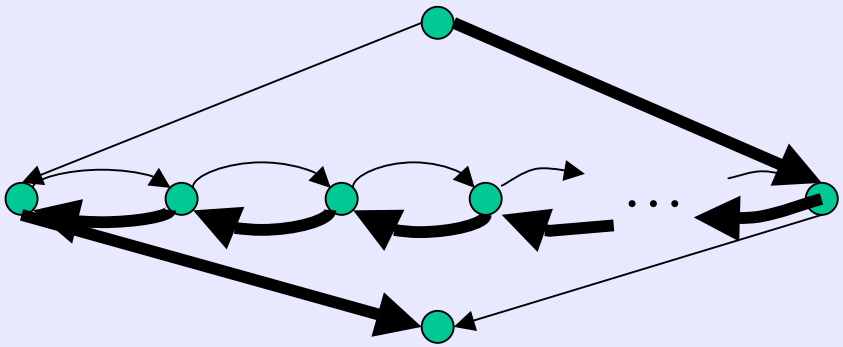
D-HAMPATH is NP-complete (6)

The path begins at **s**, goes through each diamond in turn, and ends up at **t**

To hit the horizontal nodes in a diamond, the path is either one of the following way:



Left-to-Right



Right-to-Left

D-HAMPATH is NP-complete (7)

If x_i is assigned TRUE in the satisfying assignment of F, we use Left-to-Right method to traverse the corresponding diamond. Otherwise, if x_i is assigned FALSE, we use the Right-to-Left method

So far, the path covers all the nodes in G except the clause nodes. We can easily include them by adding detours at the horizontal node

D-HAMPATH is NP-complete (8)

In each clause, we select a literal that is assigned TRUE in the satisfying assignment of F

Suppose we select x_i in clause C_m. Then, in our current path, the horizontal nodes in x_i are from Left-to-Right. Also, by our construction of edges in P. 11, we can see that we can easily detour at the m^{th} pair of horizontal nodes, and cover C_m

D-HAMPATH is NP-complete (9)

Similarly, if we select $\neg x_i$ in clause C_m, then in our current path, the horizontal nodes in x_i are from Right-to-Left. Also, by our construction of edges in P. 12, we can see that we can easily detour at the m^{th} pair of horizontal nodes, and cover C_m

Thus, if F is satisfiable, G has a Hamiltonian path from s to t (proof of the => direction done)

D-HAMPATH is NP-complete (10)

(<=) For the other direction, if G has a Hamiltonian path from s to t, we shall demonstrate a satisfying assignment for F

Firstly, we take a look at the Hamiltonian path. If it is "normal"--- that is, visiting the diamonds in the order from top one to the bottom one (excluding the detours to clause nodes) --- we can obtain a satisfying assignment as follows: (next slide)

D-HAMPATH is NP-complete (11)

- If the path is Left-to-Right in the diamond for x_i, we assign TRUE to x_i
- If the path is Right-to-Left in the diamond for x_i, we assign FALSE to x_i

Because each clause node is visited once in the Hamiltonian path, by determining how the detour is taken, we know that at least one literal in each clause is TRUE

D-HAMPATH is NP-complete (12)

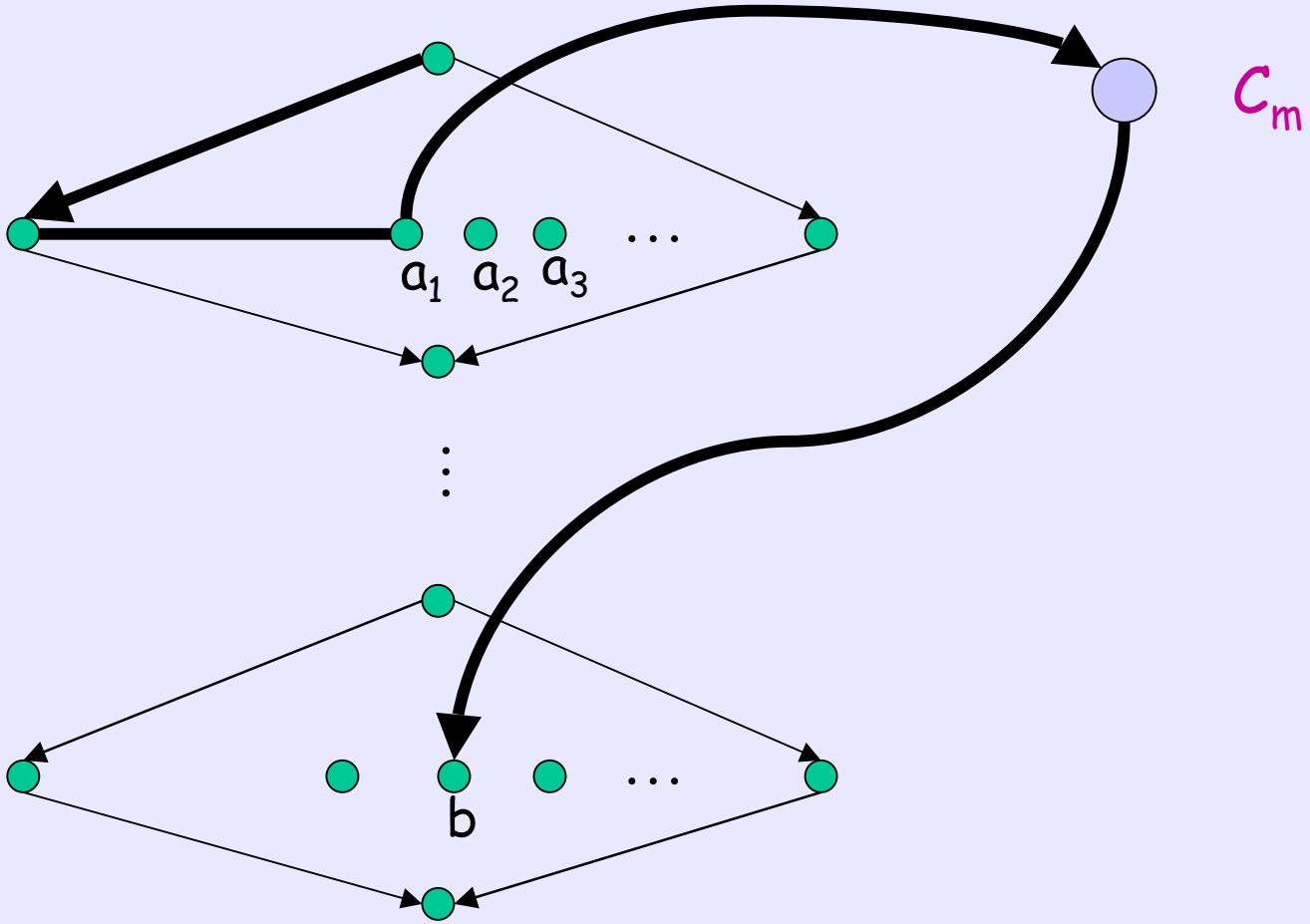
It remains to show is:

Hamiltonian path must be normal

We prove this by contradiction.

Suppose on the contrary that the path is not normal. Then, the Hamiltonian path must have entered a clause from one diamond, but left the clause to another diamond, as shown in next slide:

When Hamiltonian Path not Normal



D-HAMPATH is NP-complete (13)

The Hamiltonian path goes from a_1 to c but instead of returning to a_2, it goes to b in a different diamond

However, in our graph G, a_2 is connected to at most three nodes: a_1 , a_3, and c (why?)

If a_2 is a separator: connects to a_1 and a_3 only

Else: connects to a_1, a_3, and c only

D-HAMPATH is NP-complete (14)

In the current path, a_1 and c have both been visited now. So a_2 cannot find two distinct nodes (one incoming neighbor, one outgoing neighbor) that connects it to the current path

Thus, current path is not Hamiltonian !!

and a contradiction occurs (where?)

So, all Hamiltonian path from s to t must be normal. This implies that if such a path exists, F is satisfiable (proof of <= direction done)

D-HAMPATH is NP-complete (15)

In conclusion, we have

$$\langle F \rangle \in \text{3SAT} \Leftrightarrow \langle G, s, t \rangle \in \text{D-HAMPATH}$$

As it is easy to see that the above reduction from 3SAT to D-HAMPATH takes only polynomial time, D-HAMPATH is therefore NP-complete

Undirected HAMPATH

Let HAMPATH be the language

$\{ \langle G,s,t \rangle \mid G$ has a (undirected) Hamiltonian path from s to $t \}$

Theorem: HAMPATH is NP-complete.

HAMPATH is NP-complete

We just give a sketch of the proof:

1. First, HAMPATH is in NP (easy to check).
2. Next, to see why it is NP-complete, we reduce 3SAT to HAMPATH, using similar construction as we use in D-HAMPATH.

However, the graph is now undirected. Instead of using directed edges in the reduction in D-HAMPATH before, we …

HAMPATH is NP-complete (2)

... replace every node u in previous graph by 3 nodes, u_{in}, u_{mid}, u_{out}, in the new graph

A directed edge from u to v in the previous graph is now replaced by an undirected edge joining u_{out} and v_{in}

This completes the reduction, and we can show that this reduction works (exercise at home!) and takes polynomial time

Thus, HAMPATH is NP-complete

How about HAM-CIRCUIT ?

Let HAM-CIRCUIT be the language

$$\{ \langle G \rangle \mid G \text{ has a Hamiltonian circuit} \}$$

Theorem: HAM-CIRCUIT is NP-complete.

HAMCIRCUIT is NP-complete (2)

Proof: First, HAMCIRCUIT is in NP (easy to show). Then, we show it is NP-complete by reduction from HAMPATH.

To determine if $\langle G,s,t \rangle$ is in HAMPATH, we construct G' by adding to G a new vertex v, and two edges $\{v,s\}$ and $\{v,t\}$. Then it is easy to see that:

$\langle G,s,t \rangle \in$ HAMPATH $\Leftrightarrow \langle G' \rangle \in$ HAMCIRCUIT

SUBSET-SUM is NP-Complete

Let S be a set of positive integers.

Let SUBSET-SUM be the language

$\{ \langle S, t \rangle \mid S$ has a subset whose sum is $t \}$

Theorem: SUBSET-SUM is NP-complete.

SUBSET-SUM is NP-complete (2)

Proof: First, SUBSET-SUM is in NP (easy to show). We show it is NP-complete by reduction from 3SAT

Let F be a Boolean formula in 3cnf-form. Let $x_1, x_2, \ldots, x_j$ be its variables and let $C_1, C_2, \ldots, C_k$ be its clauses.

We transform F into a set S of $2j+2k$ (very large) numbers, with each having $j+k$ digits

(the first j digits corresponds to variables, and the following k digits corresponds to clauses)

SUBSET-SUM is NP-complete (3)

For each variable x_i, we create two numbers y_i and z_i, such that their i^{th} digit (corresponding to x_i) is set to 1. Also,

- if x_i appears in C_m, set the $(j+m)^{th}$ digit of y_i (which corresponds to c_r) to 1
- if $\neg x_i$ appears in C_m, set the $(j+m)^{th}$ digit of z_i (which corresponds to c_m) to 1

The remaining digits of y_i and z_i are set to 0

Constructing the numbers in S

	1	2	3	…	j	C_1	C_2	…	C_k
y_1	1	0	0	…	0	1	0		
z_1	1	0	0	…	0	0	0		
y_2	0	1	0	…	0	0	1		
z_2	0	1	0	…	0	1	0		
⋮									
y_j	0	0	0	…	1	0	1		
z_j	0	0	0	…	1	0	0		

Assume $C_1 = (x_1 \vee \neg x_2 \vee x_3)$ and $C_2 = (x_2 \vee \neg x_3 \vee x_j)$

SUBSET-SUM is NP-complete (4)

In addition, S contains one pair of numbers, g_m and h_m for each clause c_m

These two numbers are equal, with the $(j+m)^{th}$ digit (which corresponds to c_m) set to 1

[Later, they are used as 'fillers' to get a subset sum]

Let t = 111…1 333…3 [j 1s followed by k 3s] be the target number. We shall show:

F is satisfiable ⇔ a subset of S adds to t

SUBSET-SUM is NP-complete (5)

(=>) Suppose F is satisfiable. We select y_i if x_i is set to TRUE. Else, we select z_i

If we add the numbers selected so far,

1. Leftmost j digits are correct (why?)
2. Each of the remaining k digits is between 1 and 3 (why?)

Now, we can select suitable g_m or h_m (or both) to fill up the differences, thus getting t

SUBSET-SUM is NP-complete (6)

(<=) On the other hand, suppose a subset of S adds up to t. Then, exactly one of the y_i or z_i is in this subset (why?)

By setting x_i to TRUE when y_i is in the subset, and FALSE when z_i is in the subset

We claim F is satisfied: Consider the numbers in the subset whose $(j+m)^{th}$ bit (corresponding to c_m) is 1

SUBSET-SUM is NP-complete (7)

There must be 3 such numbers (why?), so one of them must be some y_i or z_i (why?)

- If it is y_i, it means (i) C_m contains x_i, and (ii) we have assigned x_i to TRUE
- If it is z_i, it means (i) C_m contains $\neg x_i$, and (ii) we have assigned x_i to FALSE

In both cases, C_m is satisfied

Thus, all clauses are satisfied, and so is F

SUBSET-SUM is NP-complete (8)

Now, we have shown that

$$\langle F \rangle \in \text{3SAT} \Leftrightarrow \langle S, t \rangle \in \text{SUBSET-SUM}$$

Also, it is easy to check that the above reduction takes polynomial time (in terms of the length of F).

Thus, SUBSET-SUM is NP-complete.

PARTITION is NP-Complete

Let S be a set of positive integers.

Let PARTITION be the language

{ ⟨S⟩ | S can be partitioned into two groups such that the sum in each group is the same }

Theorem: PARTITION is NP-complete.

PARTITION is NP-complete (2)

Proof: First, PARTITION is in NP (easy to show). Then, we show it is NP-complete by reduction from SUBSET-SUM.

To determine if S,k is in SUBSET-SUM, let X = sum of values in S. We construct S' by adding the two numbers 2X-k and X+k to S. Then it is easy to see that (why??):

$\langle S,k \rangle \in$ SUBSET-SUM $\Leftrightarrow \langle S' \rangle \in$ PARTITION

Brain Teaser 1: HITTING SET

Let C be a collection of subsets of S. A set of S' is called a hitting set for C if every subset of C has at least one element in S'.

Let HITTING-SET be the language

$\{ \langle C,k \rangle \mid C$ is a collection of subsets with a hitting set of size $k \}$

Theorem: HITTING-SET is NP-complete.

Brain Teaser 2: SUBGRAPH ISOMORPHISM

We say two graph $H=(V,E)$ and $H'=(V',E')$ are isomorphic if there exists a one-to-one function $f: V' \rightarrow V$ such that

$\{ u,v \}$ in E if and only if $\{ f(u),f(v) \}$ in E'

Let SUBGRAPH-ISO be the language

$\{ \langle G,H \rangle \mid G$ has a subgraph isomorphic to $H \}$

Theorem: SUBGRAPH-ISO is NP-complete

SUBGRAPH-ISO is NP-complete

Proof: First, SUBGRAPH-ISO is in NP (easy to show). Then, we show it is NP-complete by reduction from CLIQUE.

Given G,k, we construct G',H' as follows:

Set $G' = G$. Set $H' =$ k-clique. Then it is easy to see that:

$\langle G,k\rangle \in$ CLIQUE $\Leftrightarrow \langle G',H'\rangle \in$ SUBGRAPH-ISO

Brain Teaser 3: BOUNDED-DEG SPANTREE

A spanning tree of a graph $G=(V,E)$ is a tree containing every vertex in G, and whose edges are from E. A degree-k spanning tree is a spanning tree such that degree of each internal node is at most k

Let Bounded-Deg-ST be the language

$\{ \langle G,k \rangle \mid G$ has a degree-k spanning tree $\}$

Theorem: Bounded-Deg-ST is NP-complete

Bounded-Deg-ST is NP-complete

Proof: First, Bounded-Deg-ST is in NP (easy to show). Then, we show it is NP-complete by reduction from HAMPATH

Hint: What is a degree-2 spanning tree?

A degree-2 spanning tree is a Hamiltonian path in the graph!!!

Bounded-Deg-ST is NP-complete (2)

Now, given a graph G, we can transform G into G' by adding two nodes, u and v, and two edges, $\{u,s\}$ and $\{v,t\}$. Then, we can see

$$\langle G,s,t\rangle \in \text{HAMPATH}$$

$$\Leftrightarrow$$

$$\langle G',2\rangle \in \text{Bounded-Deg-ST}$$

So, Bounded-Deg-ST is NP-complete

Brain Teaser 4: KNAPSACK

Let S be a set of items, each item x in S has a positive integral value v(x) and a positive integral weight w(x).

Let KNAPSACK be the language

{ ⟨S,b,k⟩ | a subset of items in S of total weight at most b, but whose total value is at least k }

Theorem: KNAPSACK is NP-complete.

KNAPSACK is NP-complete

Proof: First, KNAPSACK is in NP (easy to show). Then, we show it is NP-complete by reduction from PARTITION

Let $S = \{s_1, s_2, \ldots, s_j\}$ be a set of +ve integers. We want to construct S', b, and k such that

$\langle S \rangle \in$ PARTITION $\Leftrightarrow \langle S', b, k \rangle \in$ KNAPSACK

The construction of S' is as follows:

KNAPSACK is NP-complete (2)

For each s_i in S, we create x_i in S' such that

$$w(x_i) = v(x_i) = s_i$$

Also, set $b = k = Y/2$, where $Y = s_1+s_2+\ldots+s_j$

Now, if S has a partition, then a subset of numbers in S adds up to $Y/2$. By choosing the items of S' that corresponds to this subset, the total weight $\leq b$ and the total value $\geq k$ (why?)

Thus, $\langle S \rangle \in$ PARTITION ➔ $\langle S', b, k \rangle \in$ KNAPSACK

KNAPSACK is NP-complete (3)

On the other hand, if a subset of items of S' have total weight $\leq b$ and total value $\geq k$, then the sum of the corresponding s_i in S will be at most b and at least k (why?).

Since $b = k = Y/2$, we have the sum of those items in $S = Y/2$

Thus, $\langle S', b, k \rangle \in$ KNAPSACK ➔ $\langle S \rangle \in$ PARTITION

As the reduction can be done in polynomial time, KNAPSACK is NP-complete.

What we have learnt

- The class P, and the class NP
- Some problems in NP are the most difficult ones in the set. We call them NP-complete problems
- SAT is NP-complete
- Other problems in NP can be shown to be NP-complete using polynomial time reduction (from what to what?)

Next Time

- Chapter 8 (not in exam) or Revision?

Decision vs Optimization

- The problems we have seen so far are called decision problems, because we want to decide if a string is in a set or not
- For instance, in the vertex cover problem, we have a language VERTEX-COVER

  $\{ \langle G,k \rangle \mid G$ is a graph with a vertex cover of size $k \}$

and our task is to decide if a string is in the set or not

Decision vs Optimization

- For most decision problems we studied, we can define its optimization version
- E.g, in vertex cover problem, we can ask:

  Given a graph G, what is the minimum subset of vertices that can cover G?

- For the above problem, we don't want a yes/no answer, but we want a specific cover whose size is minimum

Decision vs Optimization

- Recall: VERTEX-COVER is NP-complete
  - ➔ Unlikely to solve its optimization (min) version in polynomial time
- Other than give up, we may find:
  - Better exponential-time algorithm
    - improve from $O(2^n)$ to $O(1.28^n)$
  - Approximation algorithm
  - Randomized algorithm
  - Simplified version
    - restrict G to be bipartite

Approximation Algorithm?

- Algorithm with performance guarantee
- E.g., an algorithm that can always find a cover whose size is at most k times the optimal one

➔ We call this a k-approximation algorithm for the vertex cover problem

Approximation Algorithm

- In general, a k-approximation algorithm
  - for minimization problem guarantees a solution at most k times the optimal one; while
  - for maximization problem (such as finding a maximum clique in G) guarantees a solution at least 1/k times the optimal one
- smaller k = better guarantee

Approximation Algorithm

- For vertex cover, can we find a good approximation algorithm?
  - Good means:
    - It runs fast (in polynomial time)
    - It has small k
- Let us try to find a 2-approximation algorithm for vertex cover