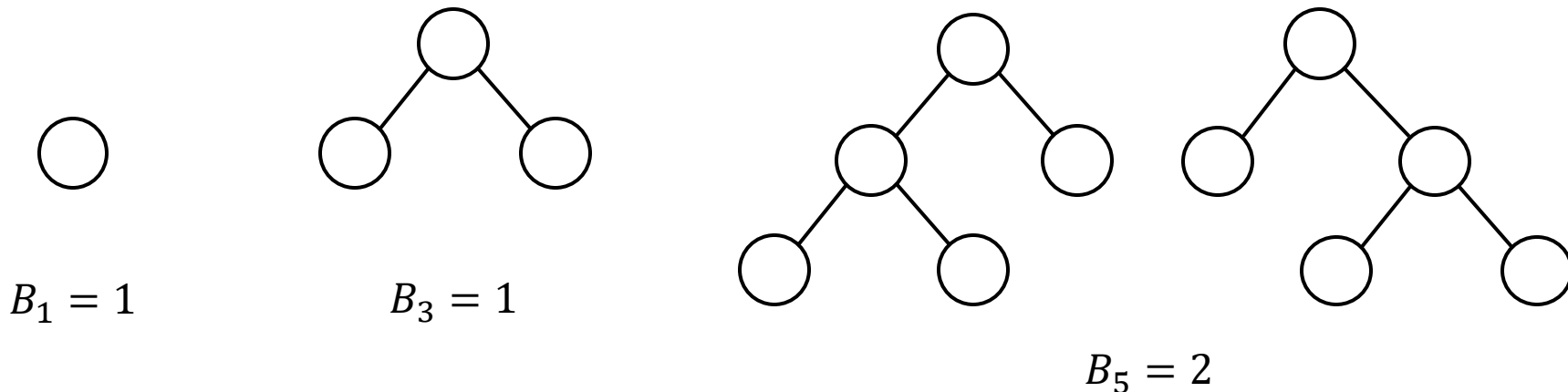


作业二

2.13. A binary tree is full if all of its vertices have either zero or two children. Let B_n denote the number of full binary trees with n vertices.

- (a) By drawing out all full binary trees with 3, 5, or 7 vertices, determine the exact values of B_3 , B_5 , and B_7 . Why have we left out even numbers of vertices, like B_4 ?



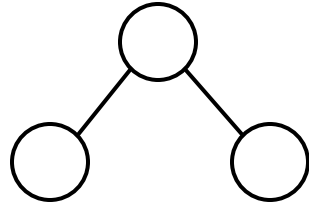
A full binary tree always has an odd number of nodes.

2.13. A binary tree is full if all of its vertices have either zero or two children. Let B_n denote the number of full binary trees with n vertices.

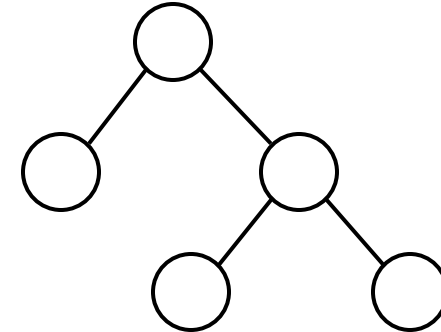
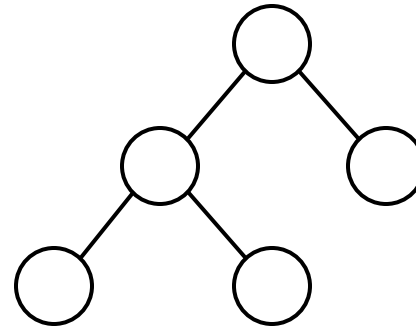
(b) For general n , derive a recurrence relation for B_n .



$$B_1 = 1$$



$$B_3 = 1$$



$$B_5 = 2$$

$$B_7 = B_1 B_5 + B_3 B_3 + B_5 B_1 = 5$$

$$B_n = B_1 B_{n-1-1} + B_3 B_{n-1-3} + \cdots + B_{n-2} B_1$$

2.13. A binary tree is full if all of its vertices have either zero or two children. Let B_n denote the number of full binary trees with n vertices.

(c) Show that B_n is $O(2^n)$.

$$\text{Catalan number: } C_n = C_1 C_{n-1} + C_2 C_{n-2} + \cdots + C_{n-1} C_1 = \frac{1}{n+1} \binom{2n}{n}$$

$$\text{According to the binomial coefficient: } \binom{n}{k} \leq 2^n$$

$$\text{For this case, } B_n = C_{\frac{n-1}{2}} = \frac{2}{n+1} \binom{\frac{n-1}{2}}{\frac{n-1}{2}} \leq 2^{n-1} < 2^n$$

2.19. A k -way merge operation. Suppose you have k sorted arrays, each with n elements, and you want to combine them into a single sorted array of kn elements.

- (a) Here's one strategy: Using the merge procedure from Section 2.3, merge the first two arrays, then merge in the third, then merge in the fourth, and so on. What is the time complexity of this algorithm, in terms of k and n ?

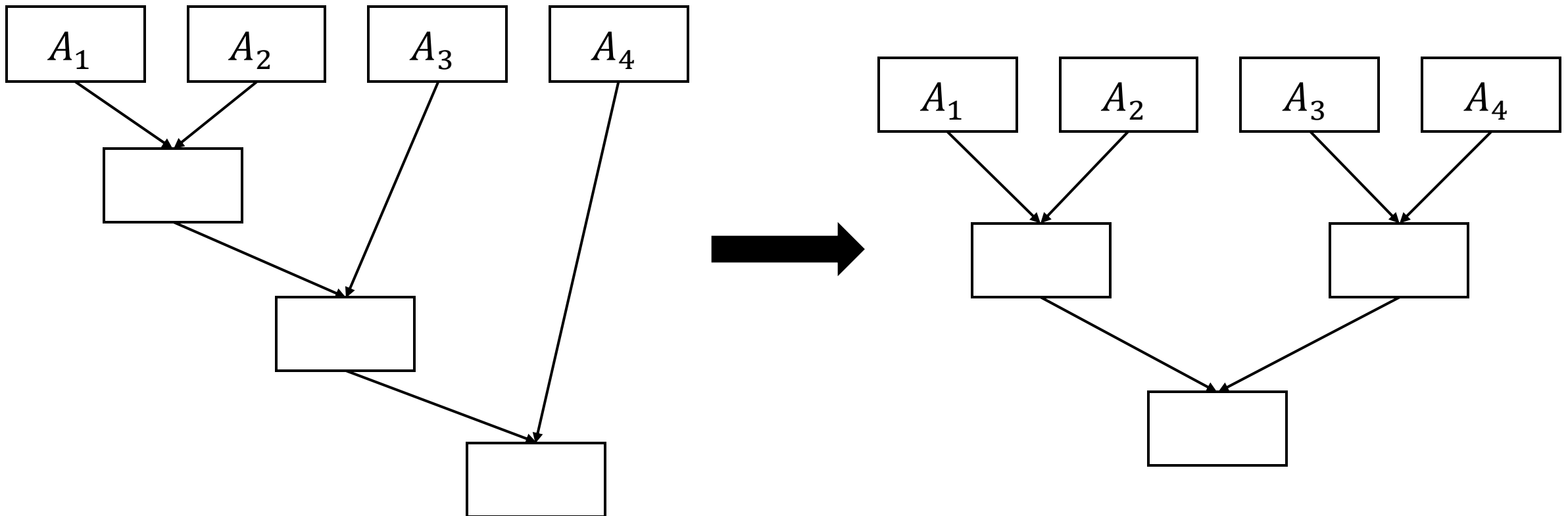
Complexity of merging two arrays of size n : $O(2n)$

In the i th step, we merge an array with $i \cdot n$ elements and another with n elements. The complexity is $O(i \cdot n + n) = O((i + 1) \cdot n)$.

$$\begin{aligned} \text{Total complexity: } O(2n + 3n + \cdots + kn) &= O((2 + 3 + \cdots + k)n) \\ &= O\left(\left(\frac{k(k+1)}{2} - 1\right)n\right) = O(n \times k^2). \end{aligned}$$

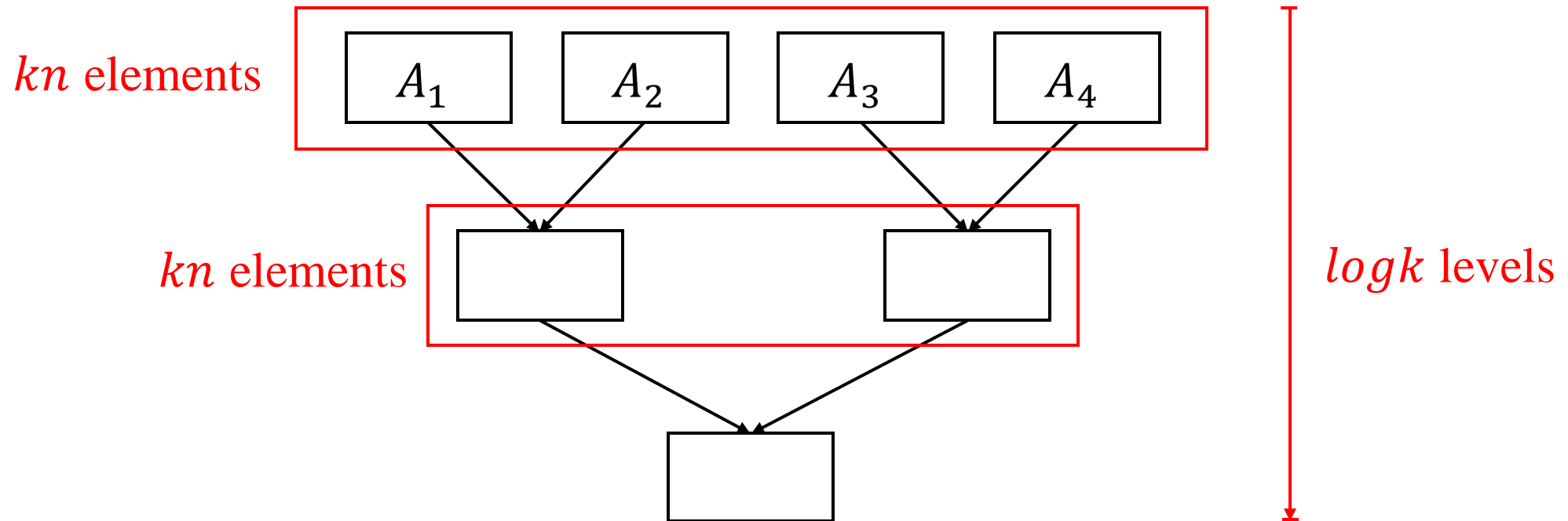
2.19. A k -way merge operation. Suppose you have k sorted arrays, each with n elements, and you want to combine them into a single sorted array of kn elements.

(b) Give a more efficient solution to this problem, using divide-and-conquer.



2.19. A k -way merge operation. Suppose you have k sorted arrays, each with n elements, and you want to combine them into a single sorted array of kn elements.

(b) Give a more efficient solution to this problem, using divide-and-conquer.



Complexity: $O(nk \log k)$

2.22. You are given two sorted lists of size m and n . Give an $O(\log m + \log n)$ time algorithm for computing the k th smallest element in the union of the two lists.

Binary Search complexity: $\log n$

$$\begin{array}{ccc} A = [2, 4, 6] & B = [1, 5, 7, 8, 10] & k = 4 \\ \uparrow & \uparrow & \\ i = 1 & j = 2 & \end{array}$$

$$\begin{array}{ccc} A = [2, 4, 6] & B = [1, 5, 7, 8, 10] & k = 4 \\ \uparrow & \uparrow & \\ i = 2 & j = 2 & \end{array}$$

3.7. A bipartite graph is a graph $G = (V, E)$ whose vertices can be partitioned into two sets ($V = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$) such that there are no edges between vertices in the same set (for instance, if $u, v \in V_1$, then there is no edge between u and v).

(a) Give a linear-time algorithm to determine whether an undirected graph is bipartite.

Breadth-First Search (BFS).

Neighboring vertices are put into two different sets.

Complexity: $O(|V| + |E|)$

3.7. A bipartite graph is a graph $G = (V, E)$ whose vertices can be partitioned into two sets ($V = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$) such that there are no edges between vertices in the same set (for instance, if $u, v \in V_1$, then there is no edge between u and v).

(b) Prove the following formulation: an undirected graph is bipartite **if and only if** it contains no cycles of odd length.

1. If a graph is bipartite, then it contains no odd cycles.

Suppose there is a cycle $C = v_1, v_2, \dots, v_k, v_1$ in a bipartite graph G , where k is odd.

Starting from $v_1 \in V_1$, alternate vertices along the cycle k times. Since k is odd, when we reach v_k , it would be in V_1 . Then, there is an edge $e = (v_k, v_1)$ in the same set, which contradicts that G is a bipartite graph.

3.7. A bipartite graph is a graph $G = (V, E)$ whose vertices can be partitioned into two sets ($V = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$) such that there are no edges between vertices in the same set (for instance, if $u, v \in V_1$, then there is no edge between u and v).

(b) Prove the following formulation: an undirected graph is bipartite **if and only if** it contains no cycles of odd length.

2. If a graph contains no odd cycles, then it is bipartite.

Two-Coloring problem: Breadth-First Search (BFS).

Since there is no odd cycle, no neighboring vertices are colored with the same color, which means it is a bipartite graph.

3.7. A bipartite graph is a graph $G = (V, E)$ whose vertices can be partitioned into two sets ($V = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$) such that there are no edges between vertices in the same set (for instance, if $u, v \in V_1$, then there is no edge between u and v).

(c) At most how many colors are needed to color in an undirected graph with exactly one odd-length cycle?

3 colors.

Remove one edge $e = (u, v)$ of the only one odd-length cycle. Since there is no odd-length cycle, the graph is bipartite and can be colored with 2 colors. Then, add the edge e again. Since the path between u and v contains even edges, u and v must be the same color. Final, we change u or v to a third color.

3.11. Design a linear-time algorithm which, given an undirected graph G and a particular edge e in it, determines whether G has a cycle containing e .

1. Remove $e = (u, v)$ from graph G .
2. Check if the endpoints u and v are still connected without edge e .
 - Depth-First Search (DFS) or Breadth-First Search (BFS)

Complexity: $O(1) + O(|V| + |E|)$

3.28. Here's an instance of 2SAT:

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

This instance has a satisfying assignment: set x_1 , x_2 , x_3 , and x_4 to true, false, false, and true, respectively.

(a) Are there other satisfying truth assignments of this 2SAT formula? If so, find them all.

2^n		x_1	x_2	x_3	x_4		x_1	x_2	x_3	x_4	
		F	F	F	F	F	F	T	T	F	F
		T	F	F	F	F	F	T	F	T	F
		F	T	T	T	F	F	F	T	T	F
		F	F	T	F	F	T	T	T	F	F
		F	F	F	T	F	T	F	T	T	F
		T	T	F	F	F	T	T	F	T	T
		T	F	T	F	F	F	T	T	T	F
	T	F	F	T	T	T	T	T	T	F	

3.28. Here's an instance of 2SAT:

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

This instance has a satisfying assignment: set x_1 , x_2 , x_3 , and x_4 to true, false, false, and true, respectively.

(a) Are there other satisfying truth assignments of this 2SAT formula? If so, find them all.

x_1	x_2	x_3	x_4		x_1	x_2	x_3	x_4	
F	F	F	F	F	F	T	T	F	F
T	F	F	F	F	F	T	F	T	F
F	T	T	T	F	F	F	T	T	F
F	F	T	F	F	T	T	T	F	F
F	F	F	T	F	T	F	T	T	F
T	T	F	F	F	T	T	F	T	T
T	F	T	F	F	F	T	T	T	F
T	F	F	T	T	T	T	T	T	F

3.28. Here's an instance of 2SAT:

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

This instance has a satisfying assignment: set x_1 , x_2 , x_3 , and x_4 to true, false, false, and true, respectively.

(a) Are there other satisfying truth assignments of this 2SAT formula? If so, find them all.

$$\begin{aligned} x_1 \vee \overline{x_2} \neq F &\rightarrow (x_1, x_2) \neq (F, T) \\ \overline{x_1} \vee \overline{x_3} \neq F &\rightarrow (x_1, x_3) \neq (T, T) \\ x_1 \vee x_2 \neq F &\rightarrow (x_1, x_2) \neq (F, F) \\ \overline{x_3} \vee x_4 \neq F &\rightarrow (x_3, x_4) \neq (T, F) \\ \overline{x_1} \vee x_4 \neq F &\rightarrow (x_1, x_4) \neq (T, F) \end{aligned}$$

x_1	x_2	x_3	x_4	x_1	x_2	x_3	x_4
F	F	F	F	F	T	T	F
T	F	F	F	F	T	F	T
F	T	T	T	F	F	T	T
F	F	T	F	T	T	T	F
F	F	F	T	T	F	T	T
T	T	F	F	T	T	F	T
T	F	T	F	F	T	T	T
T	F	F	T	T	T	T	T

3.28. Here's an instance of 2SAT:

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

This instance has a satisfying assignment: set x_1 , x_2 , x_3 , and x_4 to true, false, false, and true, respectively.

(b) Give an instance of 2SAT with four variables, and with no satisfying assignment.

$$x_1 \vee \overline{x_2} \neq F \rightarrow (x_1, x_2) \neq (F, T)$$

$$\overline{x_1} \vee \overline{x_3} \neq F \rightarrow (x_1, x_3) \neq (T, T)$$

$$x_1 \vee x_2 \neq F \rightarrow (x_1, x_2) \neq (F, F)$$

$$\overline{x_3} \vee x_4 \neq F \rightarrow (x_3, x_4) \neq (T, F)$$

$$\overline{x_1} \vee x_4 \neq F \rightarrow (x_1, x_4) \neq (T, F)$$

$$(x_1, x_4) = (T, T) \rightarrow \overline{x_1} \vee \overline{x_4} = F$$

x_1	x_2	x_3	x_4	x_1	x_2	x_3	x_4
F	F	F	F	F	T	T	F
T	F	F	F	F	T	F	T
F	T	T	T	F	F	T	T
F	F	T	F	T	T	T	F
F	F	F	T	T	F	T	T
T	T	F	F	T	T	F	T
T	F	T	F	F	T	T	T
T	F	F	T	T	T	T	T

3.28. Here's an instance of 2SAT:

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

This instance has a satisfying assignment: set x_1 , x_2 , x_3 , and x_4 to true, false, false, and true, respectively.

(c) Carry out this construction for the instance of 2SAT given above, and for the instance you constructed in (b).

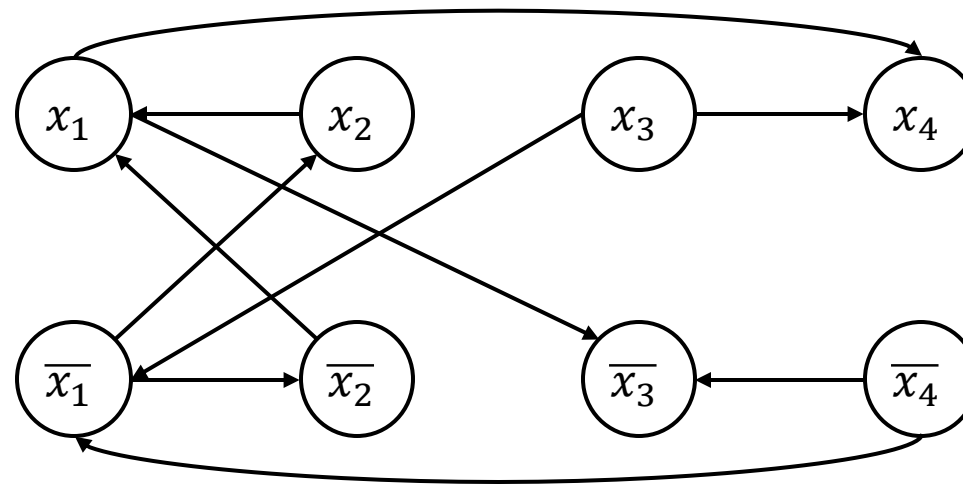
$$x_1 \vee \overline{x_2}: \overline{x_1} \Rightarrow \overline{x_2}, x_2 \Rightarrow x_1$$

$$\overline{x_1} \vee \overline{x_3}: x_1 \Rightarrow \overline{x_3}, x_3 \Rightarrow \overline{x_1}$$

$$x_1 \vee x_2: \overline{x_1} \Rightarrow x_2, \overline{x_2} \Rightarrow x_1$$

$$\overline{x_3} \vee x_4: x_3 \Rightarrow x_4, \overline{x_4} \Rightarrow \overline{x_3}$$

$$\overline{x_1} \vee x_4: x_1 \Rightarrow x_4, \overline{x_4} \Rightarrow \overline{x_1}$$



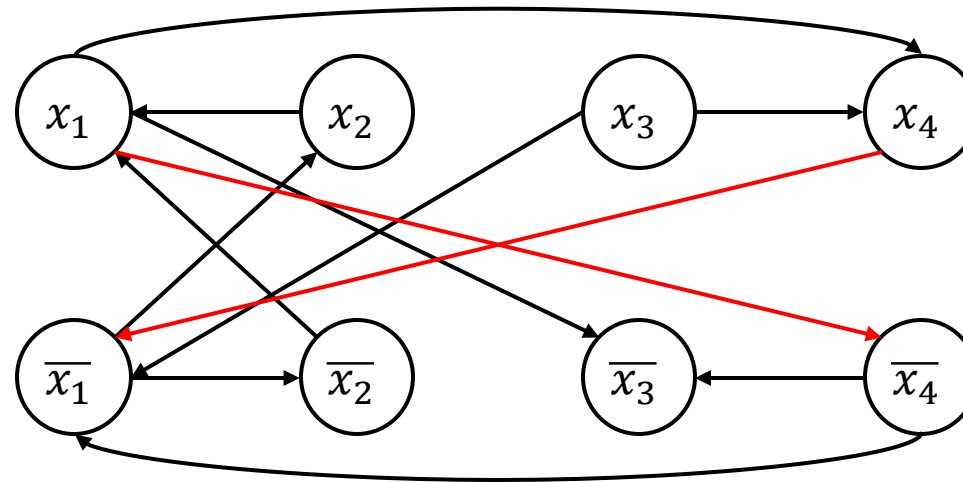
3.28. Here's an instance of 2SAT:

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

This instance has a satisfying assignment: set x_1 , x_2 , x_3 , and x_4 to true, false, false, and true, respectively.

(c) Carry out this construction for the instance of 2SAT given above, and for the instance you constructed in (b).

$$\overline{x_1} \vee \overline{x_4}: x_1 \Rightarrow \overline{x_4}, x_4 \Rightarrow \overline{x_1}$$

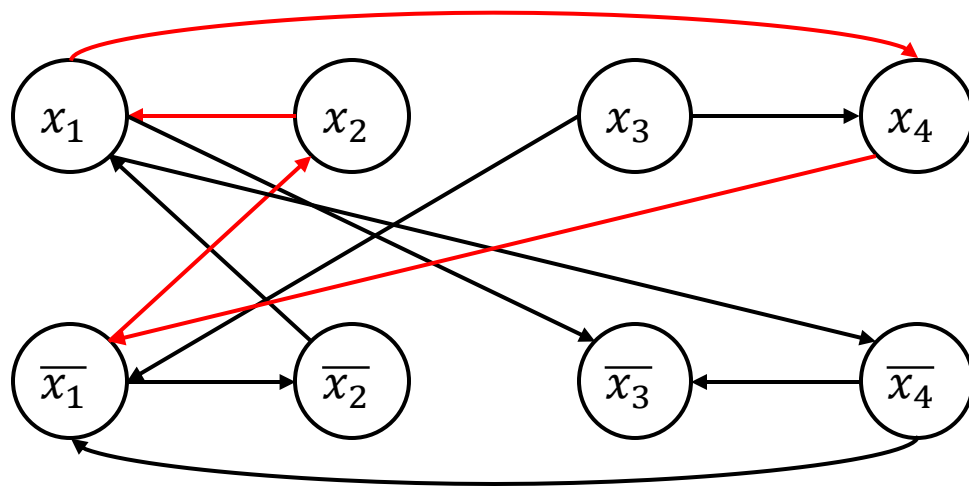


3.28. Here's an instance of 2SAT:

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

This instance has a satisfying assignment: set x_1 , x_2 , x_3 , and x_4 to true, false, false, and true, respectively.

(d) Show that if G_I has a strongly connected component containing both x and \bar{x} for some variable x , then I has no satisfying assignment.



$$(x_1 \rightarrow x_4 \rightarrow \overline{x_1}) \wedge (\overline{x_1} \rightarrow x_2 \rightarrow x_1)$$

$$\Rightarrow (x_1 \rightarrow \overline{x_1}) \wedge (\overline{x_1} \rightarrow x_1)$$

x_1	$\overline{x_1}$	$x_1 \rightarrow \overline{x_1}$	$\overline{x_1} \rightarrow x_1$	
T	F	F	T	F
F	T	T	F	F

3.28. Here's an instance of 2SAT:

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

This instance has a satisfying assignment: set x_1 , x_2 , x_3 , and x_4 to true, false, false, and true, respectively.

(e) Now show the converse of (d): namely, that if none of G_I 's strongly connected components contain both a literal and its negation, then the instance I must be satisfiable.

Since none of G_I 's strongly connected components contain both a literal and its negation, there is no contradiction that $(x \rightarrow \bar{x}) \wedge (\bar{x} \rightarrow x)$. Then we are able to find an assignment for each variable x to satisfy the instance I .

3.28. Here's an instance of 2SAT:

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

This instance has a satisfying assignment: set x_1 , x_2 , x_3 , and x_4 to true, false, false, and true, respectively.

(e) Now show the converse of (d): namely, that if none of G_I 's strongly connected components contain both a literal and its negation, then the instance I must be satisfiable.

Prove: If instance I is not satisfiable, there must be strongly connected components contain both a literal and its negation in G_I .

Since instance I is not satisfiable, there must be at least one contradiction that $(x \rightarrow \bar{x}) \wedge (\bar{x} \rightarrow x)$. According to construction rule, there must be at least one literal and its negation contained in the same strongly connected component.

3.28. Here's an instance of 2SAT:

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

This instance has a satisfying assignment: set x_1 , x_2 , x_3 , and x_4 to true, false, false, and true, respectively.

(f) Conclude that there is a linear-time algorithm for solving 2SAT.

1. Construct the implication graph G_I .

- Complexity: $O(m)$, where m is the number of clauses.

2. Find Strongly Connected Components. (Kosaraju's Algorithm)

First, perform a Depth-First Search (DFS) on the original graph and record the finishing times of each node in a stack. Then, reverse all edges in the graph. Start from the top of the stack (from the highest finishing time) and perform DFS again on the transposed graph.

- Complexity: $O(|V| + |E|)$, where $|V|$ is twice of the number of variables n and $|E| = 2m$

3.28. Here's an instance of 2SAT:

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

This instance has a satisfying assignment: set x_1 , x_2 , x_3 , and x_4 to true, false, false, and true, respectively.

(f) Conclude that there is a linear-time algorithm for solving 2SAT.

3. Check for contradictions.

- Complexity: $O(n)$

4. Construct a satisfying assignment.

We construct the component graph G_c , where each strongly connected component is a node, and there is a directed edge between two strongly connected components if there is any edge in G_I connecting vertices from those strongly connected components. Then, we topologically sort the strongly connected components. Finally, we process the strongly connected components in **reverse topological order** and assign true values to the literals.

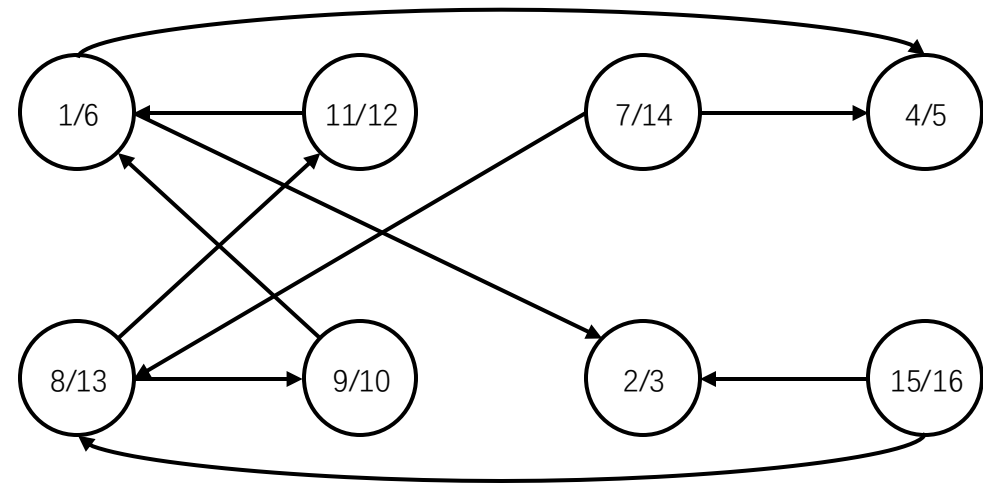
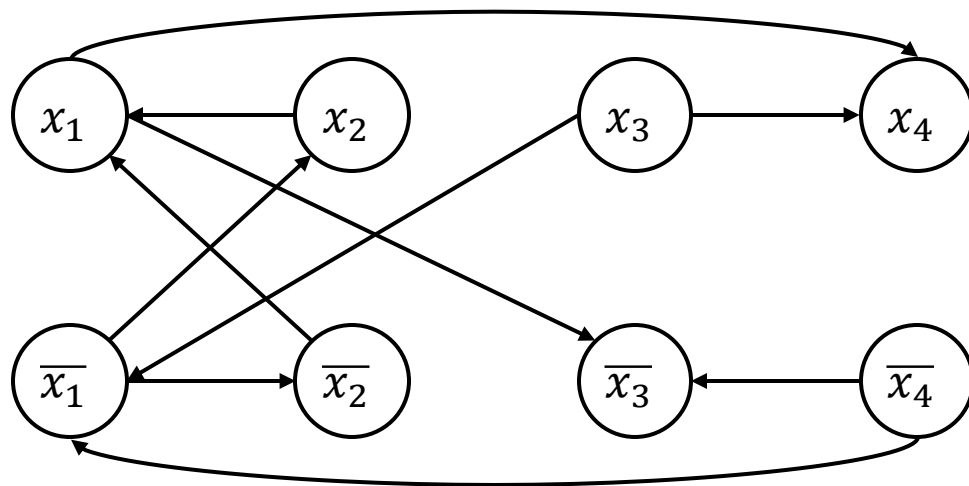
- Complexity: $O(|V| + |E|) + O(n)$

3.28. Here's an instance of 2SAT:

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

This instance has a satisfying assignment: set x_1 , x_2 , x_3 , and x_4 to true, false, false, and true, respectively.

(f) Conclude that there is a linear-time algorithm for solving 2SAT.

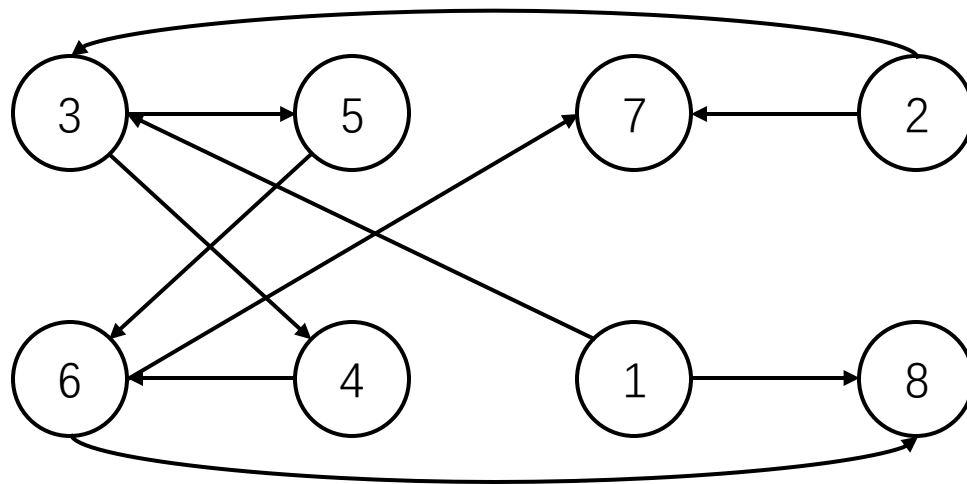


3.28. Here's an instance of 2SAT:

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

This instance has a satisfying assignment: set x_1 , x_2 , x_3 , and x_4 to true, false, false, and true, respectively.

(f) Conclude that there is a linear-time algorithm for solving 2SAT.



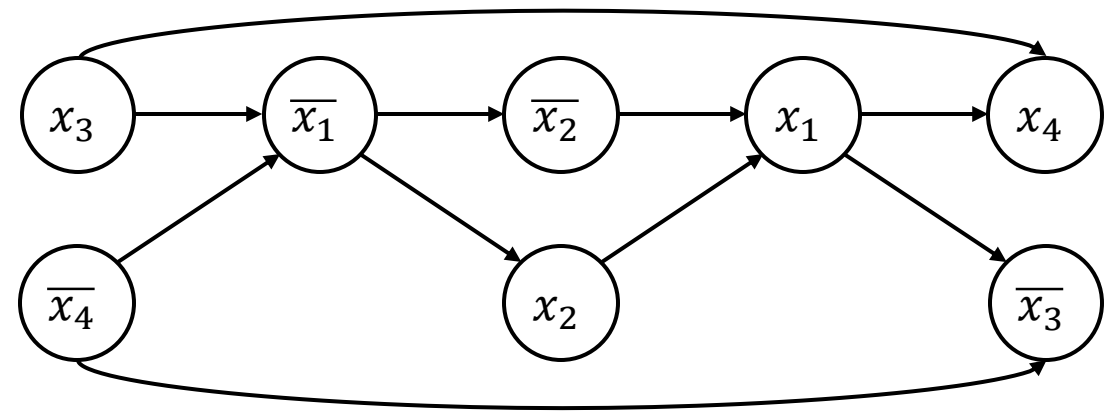
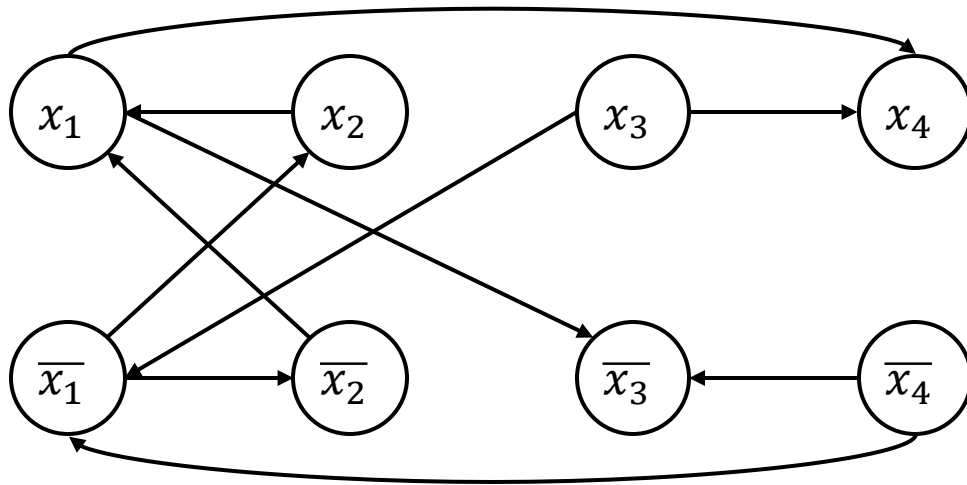
8: $\overline{x_4}$	7: x_3
6: $\overline{x_1}$	5: x_2
4: $\overline{x_2}$	3: x_1
2: x_4	1: $\overline{x_3}$

3.28. Here's an instance of 2SAT:

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

This instance has a satisfying assignment: set x_1 , x_2 , x_3 , and x_4 to true, false, false, and true, respectively.

(f) Conclude that there is a linear-time algorithm for solving 2SAT.

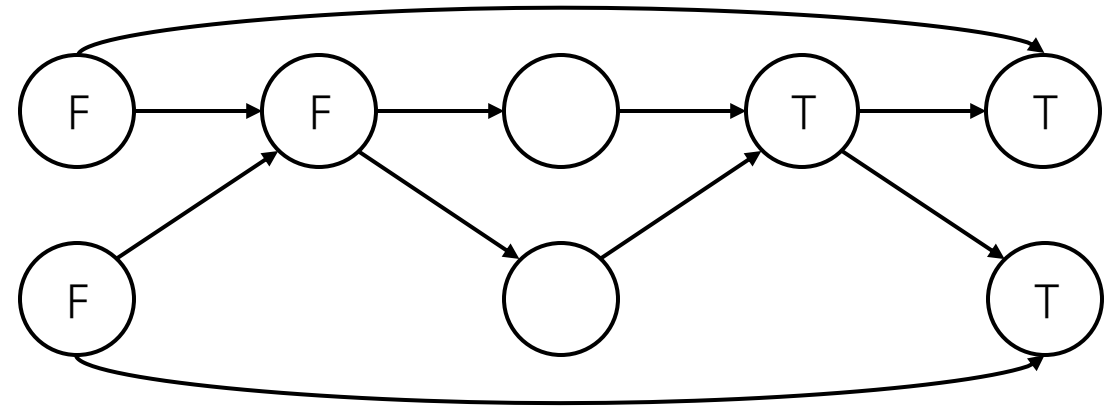
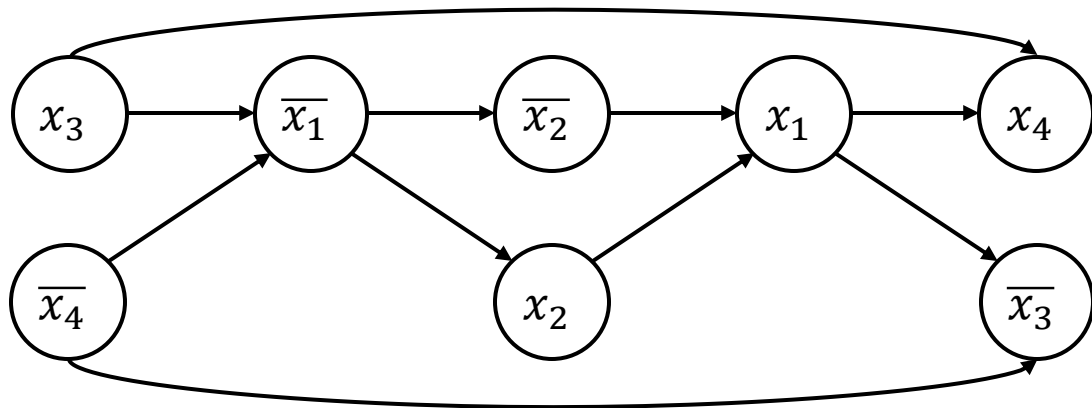


3.28. Here's an instance of 2SAT:

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

This instance has a satisfying assignment: set x_1 , x_2 , x_3 , and x_4 to true, false, false, and true, respectively.

(f) Conclude that there is a linear-time algorithm for solving 2SAT.



$$x_1 = T, x_2 = T, x_3 = F, x_4 = T$$

$$x_1 = T, x_2 = F, x_3 = F, x_4 = T$$

作业四

6.5. Pebbling a checkerboard. We are given a checkerboard which has 4 rows and n columns, and has an integer written in each square. We are also given a set of $2n$ pebbles, and we want to place some or all of these on the checkerboard (each pebble can be placed on exactly one square) so as to maximize the sum of the integers in the squares that are covered by pebbles. There is one constraint: for a placement of pebbles to be legal, no two of them can be on horizontally or vertically adjacent squares (diagonal adjacency is fine).

- (a) Determine the number of legal patterns that can occur in any column (in isolation, ignoring the pebbles in adjacent columns) and describe these patterns.

	1	2	3	4		1	2	3	4
1	0	0	0	0	5	0	0	0	1
2	1	0	0	0	6	1	0	1	0
3	0	1	0	0	7	1	0	0	1
4	0	0	1	0	8	0	1	0	1

6.5. Call two patterns compatible if they can be placed on adjacent columns to form a legal placement. Let us consider subproblems consisting of the first k columns $1 \leq k \leq n$. Each subproblem can be assigned a type, which is the pattern occurring in the last column.

(b) Using the notions of compatibility and type, give an $O(n)$ -time dynamic programming algorithm for computing an optimal placement.

	1	2	3	4	Conflict
1	0	0	0	0	
2	1	0	0	0	2, 6, 7
3	0	1	0	0	3, 8
4	0	0	1	0	4, 6
5	0	0	0	1	5, 7, 8
6	1	0	1	0	2, 4, 6, 7
7	1	0	0	1	2, 5, 6, 7, 8
8	0	1	0	1	3, 5, 7, 8

$dp[i][j]$: The maximum value that can be obtained when the first i columns are considered and the pattern of the i th column is j .

	0	1	2
1	0	$sum(pattern1)$	$\max(dp[2][1] + 0, dp[2][2] + 0, dp[2][3] + 0)$
2	0	$sum(pattern2)$	$\max(dp[2][1] + sum(pattern2), dp[2][3] + sum(pattern2))$
3	0	$sum(pattern3)$	$\max(dp[2][1] + sum(pattern3), dp[2][2] + sum(pattern3))$

$$dp[i][j] = \max(dp[i-1][k] + sum(j))$$

, where k is all patterns compatible to j .

Complexity: $O(n)$

6.10. Counting heads. Given integers n and k , along with $p_1, \dots, p_n \in [0, 1]$, you want to determine the probability of obtaining exactly k heads when n biased coins are tossed independently at random, where p_i is the probability that the i th coin comes up heads. Give an $O(n^2)$ algorithm for this task.

$dp[i][j]$: Probability of getting exactly j heads in the first i coin flips.

	0	1	2	3
0	0	$1 - p_1$	$(1 - p_1) \cdot (1 - p_2)$	$(1 - p_1) \cdot (1 - p_2) \cdot (1 - p_3)$
1	-	p_1	$p_1 \cdot (1 - p_2) + (1 - p_1) \cdot p_2$	$(1 - p_1) \cdot (1 - p_2) \cdot p_3 + (p_1 \cdot (1 - p_2) + (1 - p_1) \cdot p_2) \cdot (1 - p_3)$
2	-	-	$p_1 \cdot p_2$	$(p_1 \cdot (1 - p_2) + (1 - p_1) \cdot p_2) \cdot p_3 + p_1 \cdot p_2 \cdot (1 - p_3)$
3	-	-	-	$p_1 \cdot p_2 \cdot p_3$

$$dp[i][j] = dp[i - 1][j - 1] \cdot p_i + dp[i - 1][j] \cdot (1 - p_i)$$

Complexity: $O(n^2)$

6.17. Give an $O(nv)$ dynamic-programming algorithm for the following problem.

Input: $x_1, \dots, x_n; v$.

Question: Is it possible to make change for v using coins of denominations x_1, \dots, x_n ?

$dp[i]$: Whether we can obtain a sum of i .

0	1	2
T	T if $x_1 = 1$ or $x_2 = 1$ or ... or $x_n = 1$	T if $x_1 = 2$ or $x_2 = 2$ or ... or $x_n = 2$ or $x_2 = 2 - x_1$ or $x_2 + x_3 = 2 - x_1$ or ...

$$dp[i] = dp[i - x_1] \text{ or } dp[i - x_2] \text{ or } \dots$$

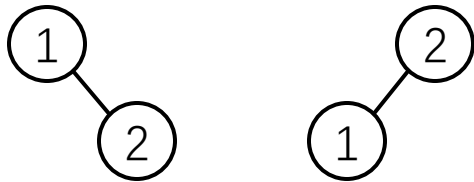
Complexity: $O(nv)$

6.20. Give an efficient algorithm for the following task.

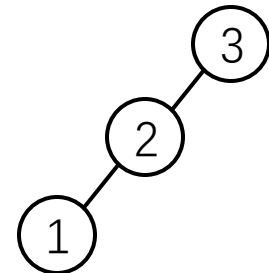
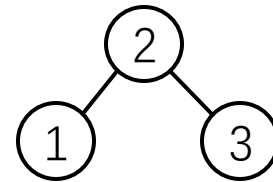
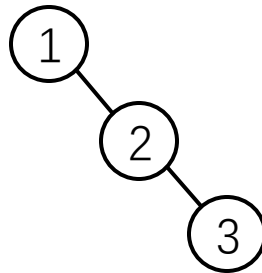
Input: n words (in sorted order); frequencies of these words: p_1, p_2, \dots, p_n .

Output: The binary search tree of lowest cost (defined above as the expected number of comparisons in looking up a word).

words = [begin, do, else, end, if, then, while]



2 words



3 words

6.20. Give an efficient algorithm for the following task.

Input: n words (in sorted order); frequencies of these words: p_1, p_2, \dots, p_n .

Output: The binary search tree of lowest cost (defined above as the expected number of comparisons in looking up a word).

words = [begin, do, else, end, if, then, while]

$dp[i][j]$: the cost of an optimal binary search tree from words[i] to words[j].

	1	2	3	4
1	0	$p_1 + p_2$	$\min(p_2 + p_3, 0 + 0, p_1 + p_2) + p_1 + p_2 + p_3$	
2	-	0	$p_2 + p_3$	$\min(p_3 + p_4, 0 + 0, p_2 + p_3) + p_2 + p_3 + p_4$
3	-	-	0	$p_3 + p_4$
4	-	-	-	0

$$dp[i][j] = \min(dp[i][k-1], dp[k+1][j]) + p_i + p_{i+1} + \dots + p_j$$

$$k = i, i+1, \dots, j$$

Complexity: $O(n^3)$

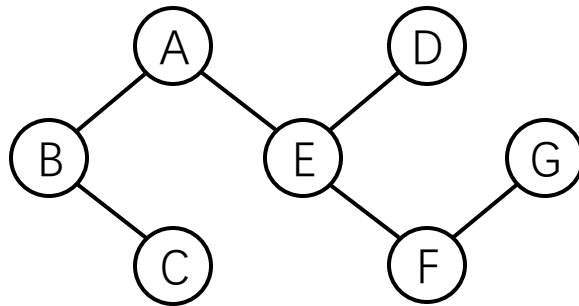
6.21. Give a linear-time algorithm for the following task.

Input: An undirected tree $T = (V, E)$.

Output: The size of the smallest vertex cover of T .

$dp[u][0]$: Size of the smallest vertex cover of the subtree rooted at u excluding u .

$dp[u][1]$: Size of the smallest vertex cover of the subtree rooted at u including u .



	G	D	C	F	E	A	B
0	0	0	0	1	$F[1] + D[1] = 2$	$E[1] = 2$	$A[1] + C[1] = 4$
1	1	1	1	1	$F[0] + D[0] + 1 = 2$	$\min(E[0], E[1]) + 1 = 3$	$\min(C[0], C[1]) + \min(A[0], A[1]) + 1 = 3$

$$dp[u][0] = dp[i][1] + dp[i + 1][1] + \dots + dp[j][1]$$

$$dp[u][1] = \min(dp[i][0], dp[i][1]) + \min(dp[i + 1][0], dp[i + 1][1]) + \dots + \min(dp[j][0], dp[j][1])$$

Complexity: $O(|V|)$

6.22. Give an $O(nt)$ algorithm for the following task.

Input: A list of n positive integers a_1, a_2, \dots, a_n ; a positive integer t .

Question: Does some subset of the a_i 's add up to t ?

$dp[i][j]$: Whether we can obtain a sum of j using the first i numbers.

	0	1	2	3
0	T	T	T	T
1	F	T if $a_1 > 1$	T if $a_1 + a_2 > 1$	T if $a_1 + a_2 + a_3 > 1$
2	F	T if $a_1 > 2$	T if $a_1 > 2$ or $a_1 > 2 - a_2$	T if $(a_1 + a_2) > 2$ or $(a_1 + a_2) > 2 - a_3$

$$dp[i][j] = dp[i - 1][j] \text{ or } dp[i - 1][j - a_i]$$

Complexity: $O(nt)$

作业六

8.3. STINGY SAT is the following problem: given a set of clauses (each a disjunction of literals) and an integer k , find a satisfying assignment in which at most k variables are true, if such an assignment exists. Prove that STINGY SAT is NP-complete.

1. Given an assignment, we can verify in **polynomial time** that at most k variables are True and that every clause is satisfied.
2. Reduction **from** SAT problem (NP-complete).

Given an instance of SAT with n variables and a collection of clauses.

We set $k = n$, then we get an instance of STINGY SAT.

If the original SAT instance is satisfiable, then there is at most n variables are true, which means the constructed STINGY SAT instance has a satisfying assignment and at most k variables are true.

If the constructed STINGY SAT instance has a satisfying assignment, then there is an assignment of values to each variable that makes every clause True, which means the original SAT instance is satisfiable.

The reduction can be finished in **polynomial time**.

8.9. In the HITTING SET problem, we are given a family of sets $\{S_1, S_2, \dots, S_n\}$ and a budget b , and we wish to find a set H of size $\leq b$ which intersects every S_i , if such an H exists. In other words, we want $H \cap S_i \neq \emptyset$ for all i . Show that HITTING SET is NP-complete.

1. Given a set, we can verify in polynomial time that its size $\leq b$ and it intersects every S_i .
2. Reduction from Vertex Cover problem (NP-complete).

Given an instance of Vertex Cover problem: $G = (V, E)$ and a budget k .

We set $S_e = \{u, v\}$ for each edge $e = (u, v) \subseteq E$ and budget $b = k$, then we get an instance of HITTING SET problem.

If a vertex cover C of size $\leq k$ such that every edge in E has at least one endpoint in C , which means there is a subset of size $\leq b$ hits every S_e .

If a hitting set H of size $\leq b$ such that at least one of u or v of every S_e is in H , which means there is a subset of size $\leq k$ covers every e .

The reduction can be finished in polynomial time.

8.14. Prove that the following problem is NP-complete: given an undirected graph $G = (V, E)$ and an integer k , return a clique of size k as well as an independent set of size k , provided both exist.

1. Given two subsets of vertices, each of size k , we can verify in polynomial time that all vertices in one set are mutually adjacent (clique) and no two vertices in another set share an edge (independent set).
2. Reduction from CLIQUE problem (NP-complete).

Given an instance of CLIQUE problem: $G = (V, E)$ and an integer b .

We set $k = b$ and add $|V|$ new vertices to G , then we get an instance of required problem G' .

If there is a clique of size b in G , then there is still a clique of size k in G' . And as $|V| > k$, we can pick any k distinct new vertices to form an independent set.

If a clique of size k as well as an independent set of size k in G' , since added vertices are all independent, the clique must lie entirely among the original vertices, which means there is a clique of size b in G .

The reduction can be finished in polynomial time.

8.15. Show that MAXIMUM COMMON SUBGRAPH is NP-complete.

Input: Two graphs $G1 = (V1, E1)$ and $G2 = (V2, E2)$; a budget b .

Output: Two set of nodes $V1' \subseteq V1$ and $V2' \subseteq V2$ whose deletion leaves at least b nodes in each graph, and makes the two graphs identical.

1. Given two subsets of nodes and a reflection of nodes in two graph, we can verify in polynomial time that two remaining subgraphs are of size $\geq b$ and are identical.
2. Reduction from CLIQUE problem (NP-complete).

Given an instance of CLIQUE problem: $G = (V, E)$ and a budget k .

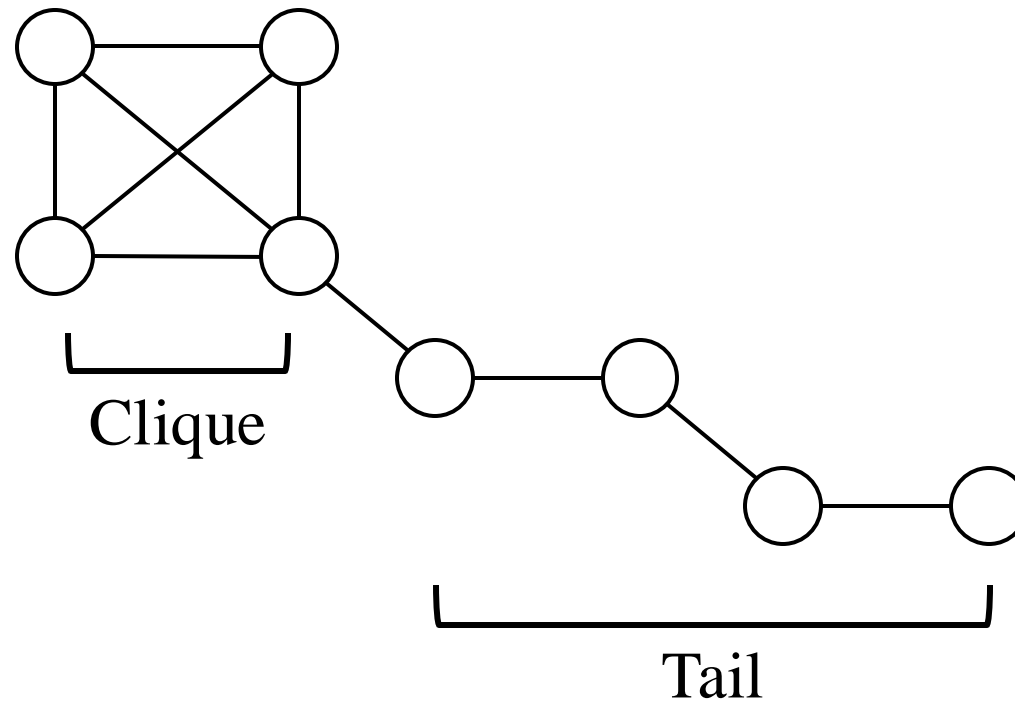
We set $b = k$ and create a complete graph with $|V|$ vertices $G2$, then we get an instance of MAXIMUM COMMON SUBGRAPH problem, $G1 = (V1, E1) = G = (V, E)$ and $G2 = (V2, E2)$.

If there is a clique of size k in G , let $V1'$ be the set of all vertices not in the clique in $G1$ and $V2'$ be any $|V| - k$ vertices in $G2$. The two remaining k vertices in $G1$ and $G2$ forms two complete subgraphs of size b , which are identical.

If two identical remaining subgraphs of size b in $G1$ and $G2$, since $G2$ is a complete graph, any remaining subgraph in $G2$ is still a complete graph, which means there is a complete subgraph (clique) of size k in $G1$.

The reduction can be finished in polynomial time.

8.19. A kite is a graph on an even number of vertices, say $2n$, in which n of the vertices form a clique and the remaining n vertices are connected in a “tail” that consists of a path joined to one of the vertices of the clique.



8.19. Given a graph and a goal g , the KITE problem asks for a subgraph which is a kite and which contains $2g$ nodes. Prove that KITE is NP-complete.

1. Given a subsets of vertices, each of size $2g$, we can verify in polynomial time that there are exactly g vertices all pairwise adjacent and the remaining g vertices form a path of length $g - 1$.
2. Reduction from CLIQUE problem (NP-complete).

Given an instance of CLIQUE problem: $G = (V, E)$ and an integer k .

We set $g = k$ and add a path of length $g - 1$ to G , connect the endpoint u_1 to each $v_i \in V$, then we get an instance of required problem G' .

If there is a clique of size k in G , there is a clique of size g in G' and it is connected to the added path of length $g - 1$, which forms a kite of size $2g$.

If there is a kite of size $2g$ in G' , since vertices in added path can not form a clique, the clique of size g must lie entirely among the original vertices, which means there is a clique of size k in G .

The reduction can be finished in polynomial time.

8.21. The reconstruction problem is now easy to state: given a multiset of k -length strings, find a string x such that $\Gamma(x)$ is exactly this multiset.

Multiset: {"ATG", "TGG", "GGC", "GCG", "CGT"}



reconstruct

$\Gamma(x) = \text{"ATGGCGT"}$

8.21. The reconstruction problem is now easy to state: given a multiset of k -length strings, find a string x such that $\Gamma(x)$ is exactly this multiset.

(a) Show that the reconstruction problem reduces to RUDRATA PATH.

Given an instance of reconstruction problem: a multiset of k -length strings $\{s_1, s_2, \dots, s_n\}$.

We set each s_i to a vertex v_i and add directed edge from v_i to v_j if the last $k - 1$ characters of s_i is equal to the first $k - 1$ characters of s_j , then we get an instance of RUDRATA PATH problem $G = (V, E)$.

If there is original string x reconstructed from $\{s_1, s_2, \dots, s_n\}$, there is a order to concatenate every string and each string appears in x exactly once, which means the concatenating order is a RUDRATA PATH in G .

If there is a RUDRATA PATH in G , there is a path visiting every vertex exactly once, which means we can reconstruct original string x by concatenating first characters of each vertex in the visiting order.

8.21. The reconstruction problem is now easy to state: given a multiset of k -length strings, find a string x such that $\Gamma(x)$ is exactly this multiset.

(b) But in fact, there is much better news. Show that the same problem also reduces to EULER PATH.

Given an instance of reconstruction problem: a multiset of k -length strings $\{s_1, s_2, \dots, s_n\}$.

We set the first $k - 1$ characters and the last $k - 1$ characters in s_i to vertices v_i and u_i . Then we add directed edge from v_i to u_i , which means we set each s_i to an edge e_i , and get an instance of EULER PATH problem $G = (V, E)$.

If there is original string x reconstructed from $\{s_1, s_2, \dots, s_n\}$, there is a order to concatenate every string and each string appears in x exactly once, which means the concatenating order is a EULER PATH in G .

If there is a EULER PATH in G , there is a path visiting every edge exactly once, which means we can reconstruct original string x by concatenating first characters of each edge in the visiting order.

作业七

9.4. Given an undirected graph $G = (V, E)$ in which each node has degree $\leq d$, show how to efficiently find an independent set whose size is at least $1/(d + 1)$ times that of the largest independent set.

Greedy Algorithm:

1. Initialize $I = \emptyset$.
2. While $V \neq \emptyset$:
3. Pick any vertex $v \in V$.
4. Add v to I .
5. Remove v and all its neighbors from V .

$$\begin{aligned} |I_{OPT}| &\leq k(d + 1) = |I|(d + 1) \\ |I| &\geq \frac{1}{d + 1} |I_{OPT}| \end{aligned}$$

9.8. In the MAX SAT problem, we are given a set of clauses, and we want to find an assignment that satisfies as many of them as possible.

(a) Show that if this problem can be solved in polynomial time, then so can SAT.

Given a SAT instance $(x_1 \vee y_1) \wedge (x_2 \vee y_2) \wedge \cdots \wedge (x_n \vee y_n)$, we use polynomial-time algorithm for MAX SAT on it.

If the maximum number of clauses that can be simultaneously satisfied is equal to the total number of clauses in the instance, then the original SAT instance is satisfiable.

Otherwise, the original SAT instance is not satisfiable.

9.8. In the MAX SAT problem, we are given a set of clauses, and we want to find an assignment that satisfies as many of them as possible.

(b) Here's a very naive algorithm.

for each variable:

set its value to either 0 or 1 by flipping a coin

Suppose the input has m clauses, of which the j th has k_j literals. Show that the expected number of clauses satisfied by this simple algorithm is

$$\sum_{j=1}^m \left(1 - \frac{1}{2^{k_j}}\right) \geq \frac{m}{2}$$

In other words, this is a 2-approximation in expectation! And if the clauses all contain k literals, then this approximation factor improves to $1 + 1/(2^k - 1)$.

9.8. In the MAX SAT problem, we are given a set of clauses, and we want to find an assignment that satisfies as many of them as possible.

(b) Show that the expected number of clauses satisfied by this simple algorithm is $\sum_{j=1}^m 1 - \frac{1}{2^{k_j}} \geq \frac{m}{2}$

$$X_j = \begin{cases} 1, & \text{if head is up} \\ 0, & \text{if head is down} \end{cases}$$

$$P(j\text{th clause is not satisfied}) = \frac{1}{2^{k_j}}, P(j\text{th clause is satisfied}) = 1 - \frac{1}{2^{k_j}}$$

$$E = \sum_{j=1}^m \left(1 - \frac{1}{2^{k_j}}\right) \geq \sum_{j=1}^m \left(1 - \frac{1}{2}\right) = \frac{m}{2} \geq \frac{OPT}{2}$$

9.8. In the MAX SAT problem, we are given a set of clauses, and we want to find an assignment that satisfies as many of them as possible.

(b) Show that if the clauses all contain k literals, then this approximation factor improves to $1 + 1/(2^k - 1)$.

$$E = \sum_{j=1}^m \left(1 - \frac{1}{2^k}\right) = m \left(1 - \frac{1}{2^k}\right) \geq OPT \left(1 - \frac{1}{2^k}\right)$$

$$\alpha = \max \frac{OPT}{E} = 1 + \frac{1}{2^k - 1}$$

9.8. In the MAX SAT problem, we are given a set of clauses, and we want to find an assignment that satisfies as many of them as possible.

(c) Can you make this algorithm deterministic?

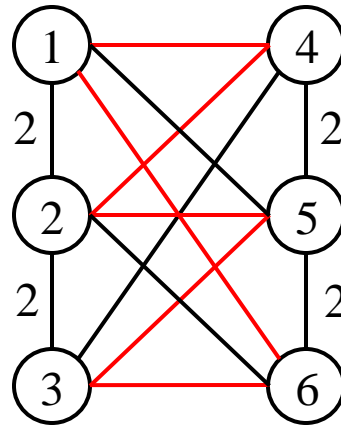
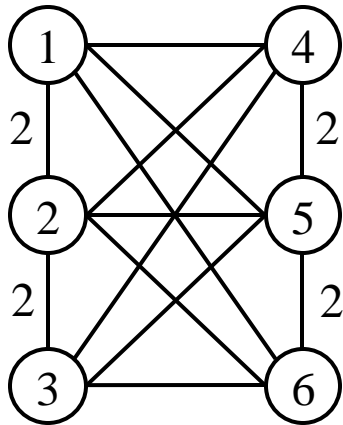
Greedy Algorithm:

1. Label the variables x_1, x_2, \dots, x_n .
2. While unsatisfied clauses $\neq \emptyset$:
3. For each x_i :
4. Pick the truth value (0 or 1) for x_i that satisfies the largest number of these currently unsatisfied clauses.

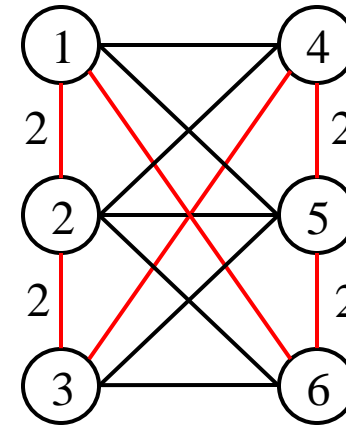
$$E_i = \max(E(X_i = 0), E(X_i = 1)) \geq \frac{E(X_i = 0) + E(X_i = 1)}{2}$$
$$E \geq \sum_{j=1}^m \left(1 - \frac{1}{2^{k_j}}\right) \geq \frac{m}{2}$$

9.10. Let us call a local search algorithm exact when it always produces the optimum solution.

(a) Show that the 2-change local search algorithm for the TSP is not exact.



$$\text{Cost} = 6 + 6 + 6 + 6 + 6 + 6 = 36$$

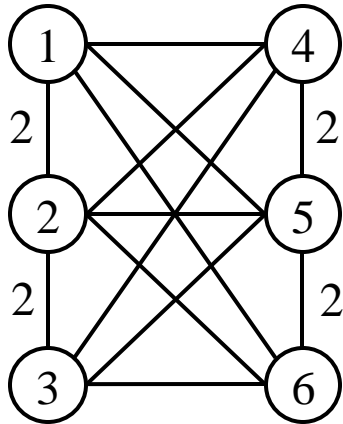


$$\text{Cost} = 2 + 2 + 6 + 2 + 2 + 6 = 20$$

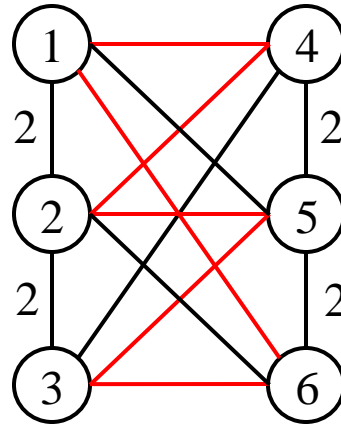
4-change local search algorithm is required.

9.10. Let us call a local search algorithm exact when it always produces the optimum solution.

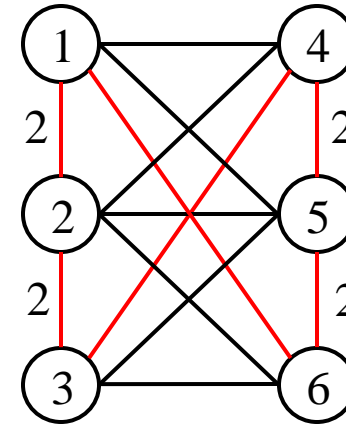
(b) Repeat for the $\lceil \frac{n}{2} \rceil$ -change local search algorithm, where n is the number of cities.



$$\text{Cost} = 6 + 6 + 6 + 6 + 6 + 6 = 36$$



$$\text{Cost} = 2 + 2 + 6 + 2 + 2 + 6 = 20$$



4-change local search algorithm is required.

9.10. Let us call a local search algorithm exact when it always produces the optimum solution.

(c) Show that the $(n - 1)$ -change local search algorithm is exact.

Prove: In complete graph, two Hamiltonian cycles have at least one common edge.

For a complete graph $G = (V, E)$, if there is no common edge in the two Hamiltonian cycles C_1 and C_2 , then there is $2n$ different edges.

Since $2n > |V| = \frac{n(n-1)}{2}$, when $n \geq 4$, there is not enough edges in G to construct two Hamiltonian cycles without common edges. When $n = 3$, there is only one Hamiltonian cycle in G .

Since two Hamiltonian cycles have at least one common edge in a complete graph, we can jump from any tour to the optimum tour in one $(n - 1)$ -change.

9.10. Let us call a local search algorithm exact when it always produces the optimum solution.

(d) Prove TSP IMPROVEMENT is NP-complete.

1. Given a new tour, we can verify in polynomial time that it is a Hamiltonian cycle and its cost is less than the original tour.
2. Reduction from TSP problem (NP-complete).

Given an instance of TSP problem: $G = (V, E)$ and a budget k .

If there is a Hamiltonian cycle in G of budget $\leq k$, we set the cost of original tour $b > k$, there is better tour with cost less than b .

If we set the cost of original tour $b = k$, there is better tour with cost less than b , there is a Hamiltonian cycle in G of budget $\leq k$.

The reduction can be finished in polynomial time.

9.10. Let us call a local search algorithm exact when it always produces the optimum solution.

(d) Prove SET COVER IMPROVEMENT is NP-complete.

1. Given a set cover, we can verify in polynomial time that it covers all elements.
2. Reduction from SET COVER problem (NP-complete).

Given an instance of SET COVER problem: A universe U , a collection $S = \{S_1, S_2, \dots, S_m\}$ and a budget k .

If there is set cover of size k that covers all elements, we set the size of original set cover $b > k$, there is a smaller set cover of size $< b$.

If we set the size of original set cover $b = k$, there is smaller set cover of size $< b$, there is a set cover of size $\leq k$.

The reduction can be finished in polynomial time.

9.10. Let us call a local search algorithm exact when it always produces the optimum solution.

(e) Show that, unless $P = NP$, there is no exact local search algorithm with polynomial iteration for the TSP and SET COVER problems.

Considering the local search algorithms for TSP and SET COVER problems, if there were an exact local search algorithm whose iterations run in polynomial time, it would imply that the TSP IMPROVEMENT and SET COVER IMPROVEMENT problems are in P . However, since these improvement problems can be reduced from the original TSP and SET COVER problems, it would then follow that TSP and SET COVER problems themselves are in P , contradicting their NP-completeness (unless $P = NP$).