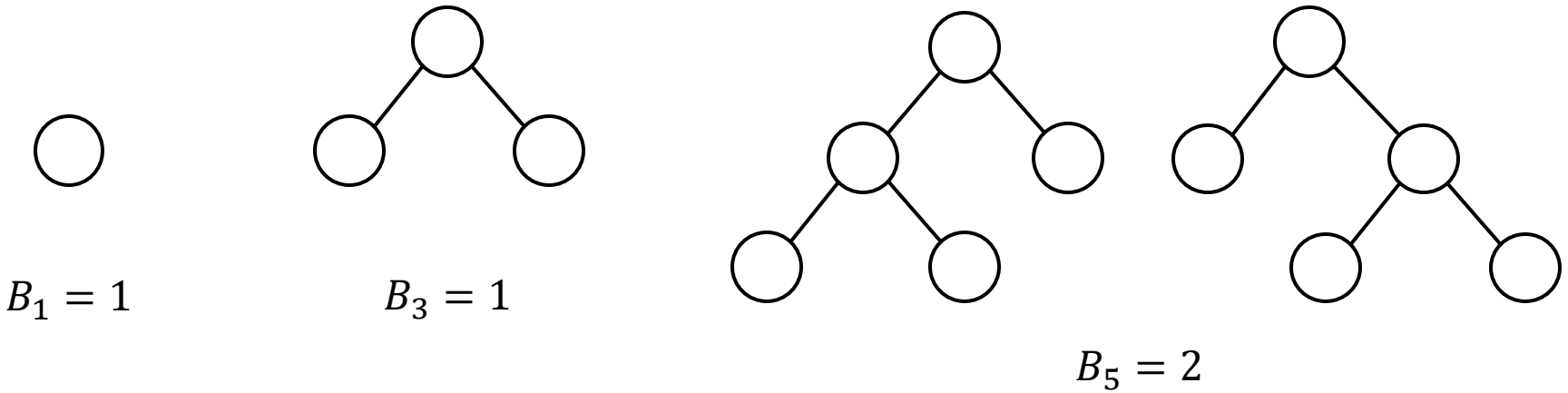
作业二

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?

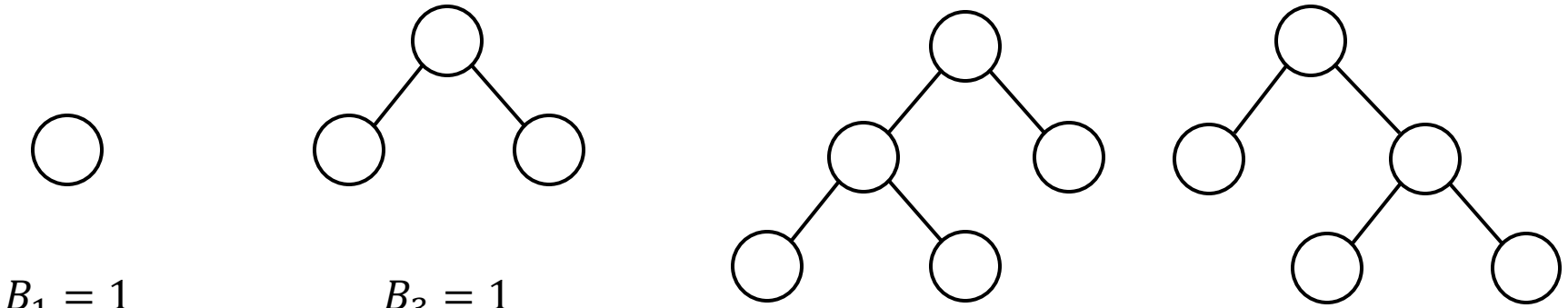


$B_1 = 1$ $B_3 = 1$ $B_5 = 2$

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.



$B_1 = 1$ $B_3 = 1$ $B_5 = 2$

$$B_7 = B_1B_5 + B_3B_3 + B_5B_1 = 5$$

$$B_n = B_1B_{n-1-1} + B_3B_{n-1-3} + \cdots + B_{n-2}B_1$$

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 $O(2^n)$.

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

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

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

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: $O(2n)$

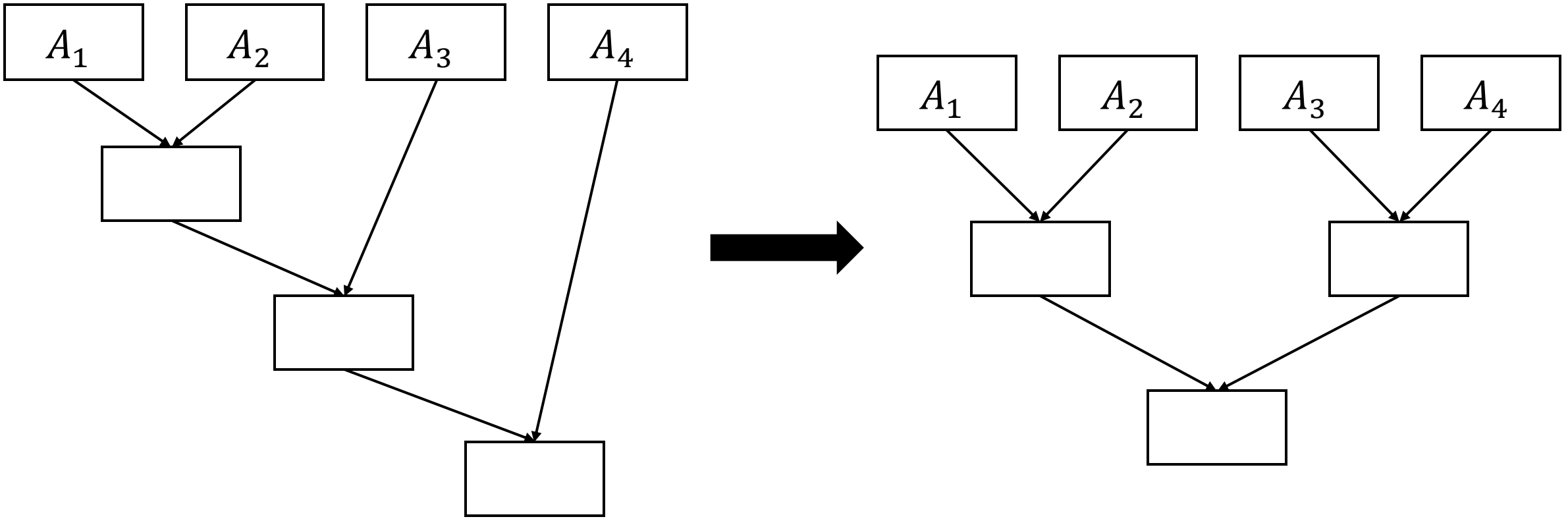
In the ith step, we merge an array with $i \cdot n$ elements and another with n elements. The complexity is $O(i \cdot n + n) = O((i + 1) \cdot n)$.

Total complexity: $O(2n + 3n + \cdots + kn) = O\big((2 + 3 + \cdots + k)n\big)$

$$= O\left(\left(\frac{k(k+1)}{2} - 1\right)n\right) = O(n \times k^2).$$

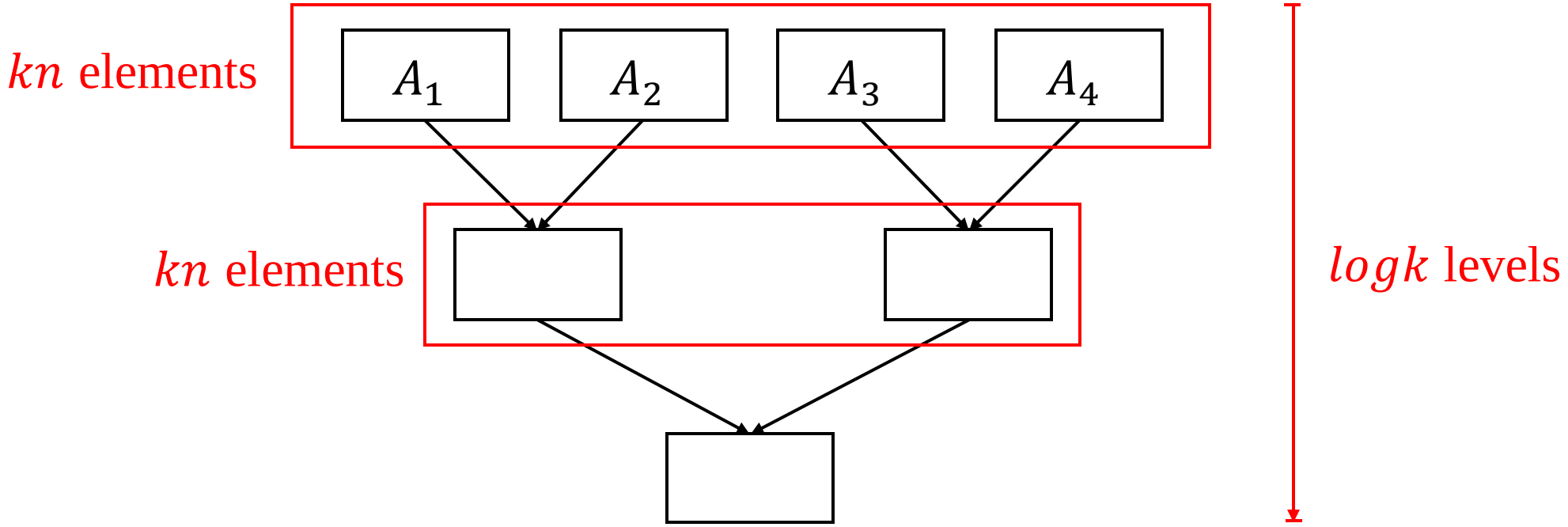
2.19. A k-way merge operation. Suppose you have k sorted arrays, each with n elements, and you want to combine them into a single sorted array of kn elements.

(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: $O(nklogk)$

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 kth smallest element in the union of the two lists.

Binary Search complexity: $\log n$

$A = [2,4,6]$ $B = [1,5,7,8,10]$ $k = 4$

$i = 1$ $j = 2$

$A = [2,4,6]$ $B = [1,5,7,8,10]$ $k = 4$

$i = 2$ $j = 2$

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 $C = v_1, v_2, \ldots, v_k, v_1$ 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.

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

x_1	x_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).

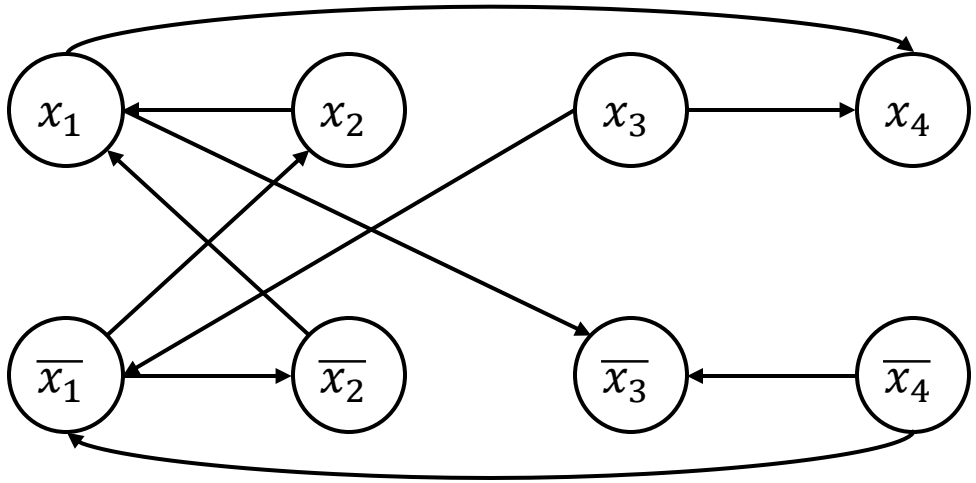
$x_1 \vee \overline{x_2}$: $\overline{x_1} \Rightarrow \overline{x_2}, x_2 \Rightarrow x_1$

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

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

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

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



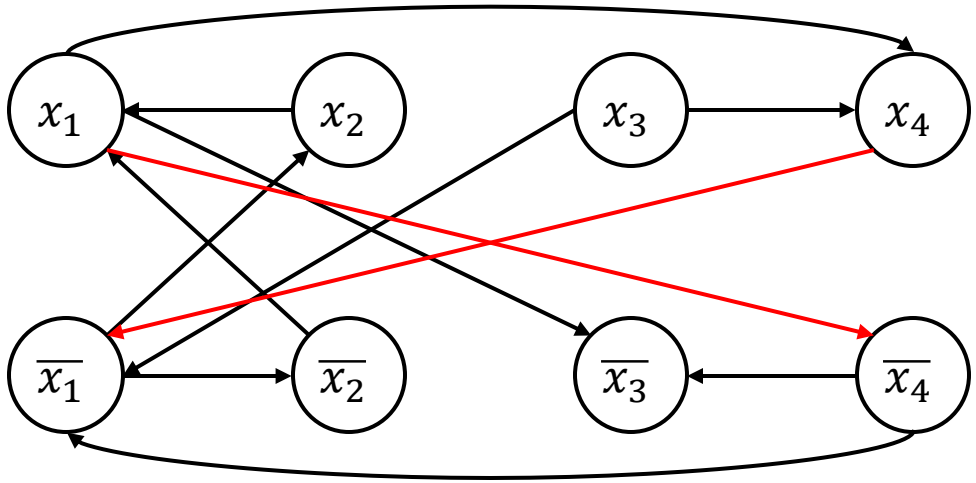
3.28. Here's an instance of 2SAT:

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

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).

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

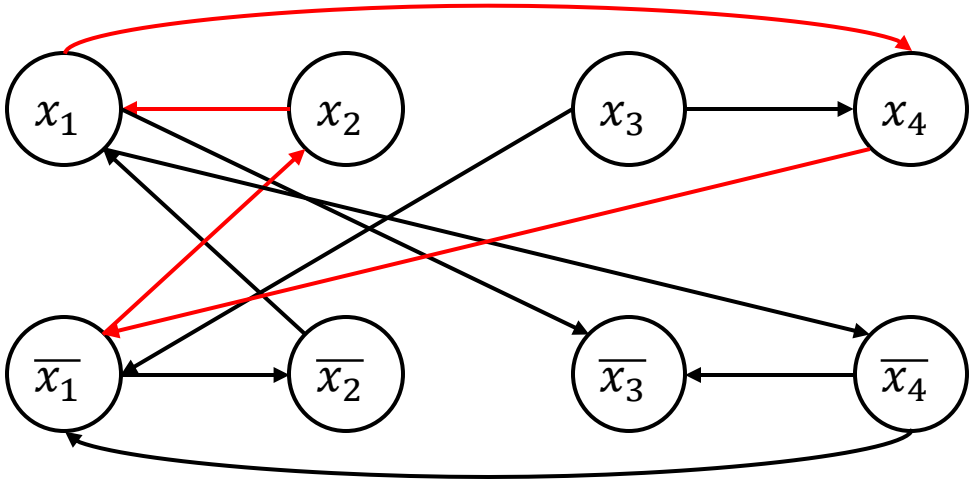


3.28. Here's an instance of 2SAT:

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

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.



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

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

x_1	$\overline{x_1}$	$x_1 \to \overline{x_1}$	$\overline{x_1} \to x_1$	
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 contradictions that $(x \to \bar{x}) \wedge (\bar{x} \to x)$. 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 $(x \to \bar{x}) \wedge (\bar{x} \to x)$. 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 sorting the strongly connected components. Final, we process the strongly connected components in reverse topological order and assign true values to the literals.

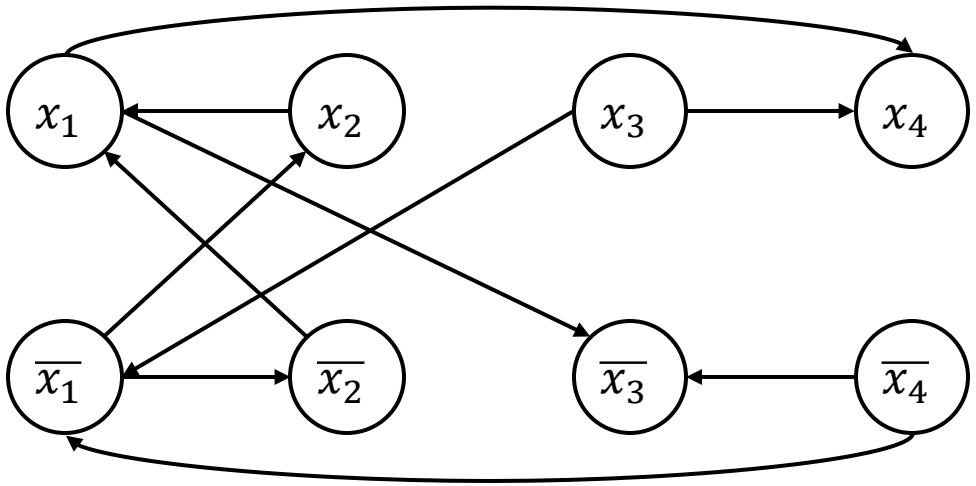
- Complexity: $O(|V|+|E|) + O(n)$

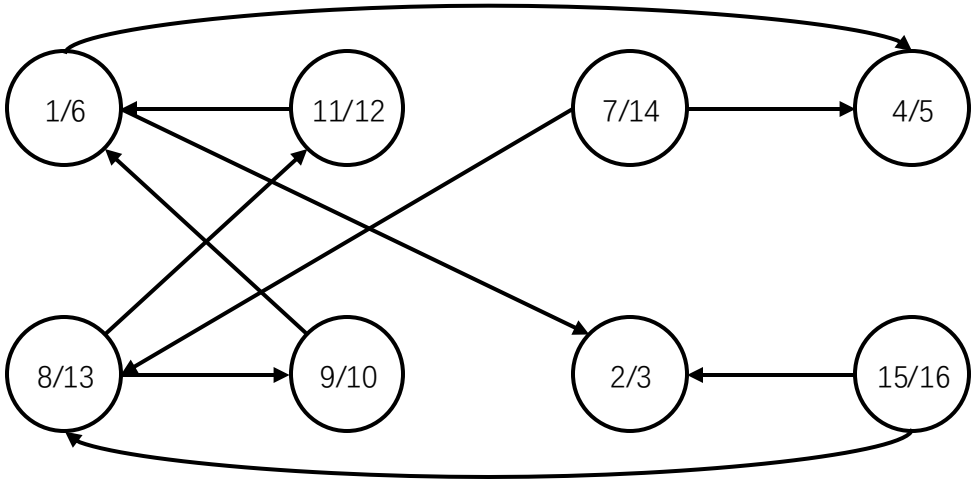
3.28. Here's an instance of 2SAT:

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

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.



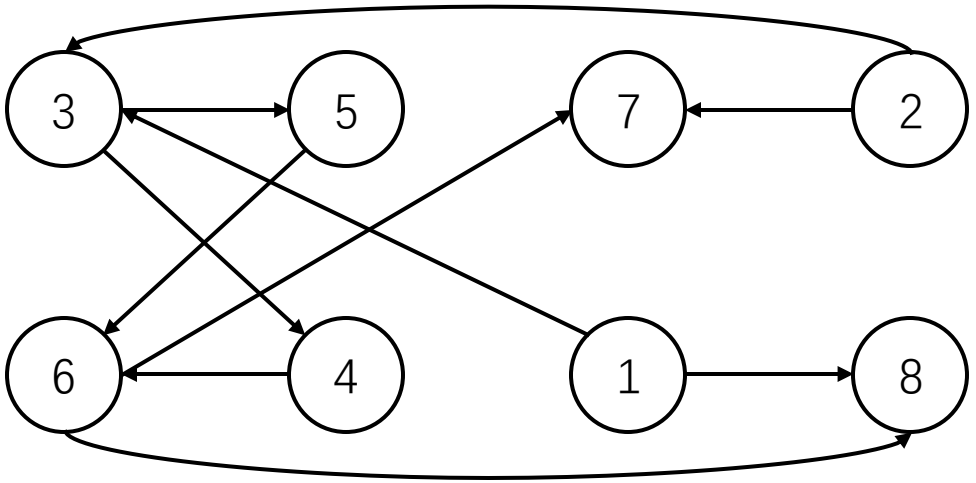


3.28. Here's an instance of 2SAT:

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

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

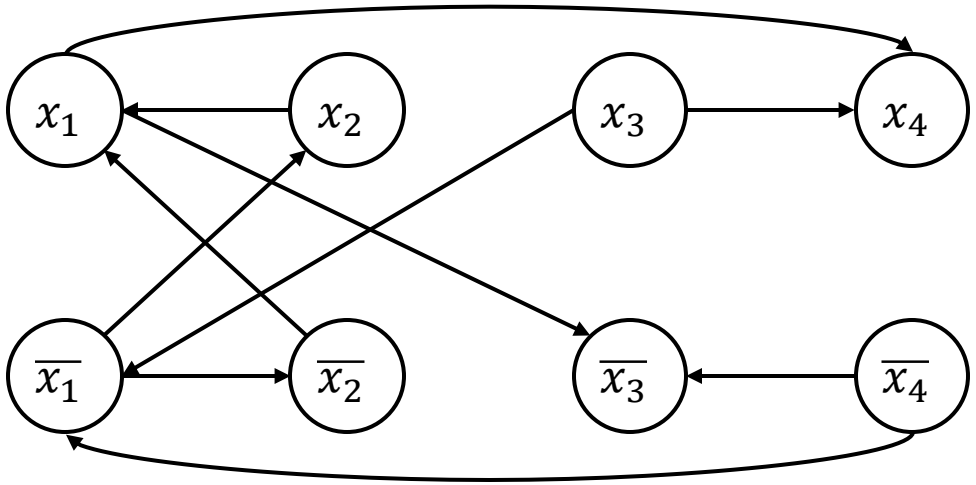
2: x_4 1: $\overline{x_3}$

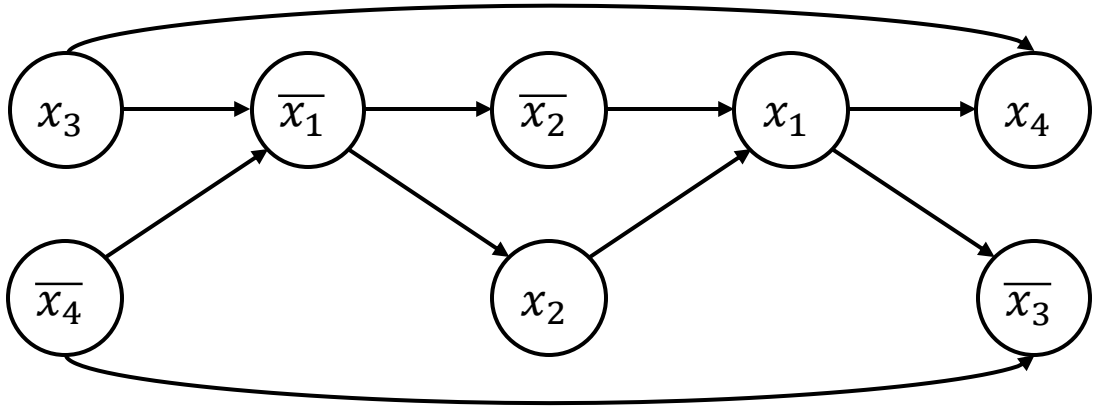
3.28. Here's an instance of 2SAT:

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

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.



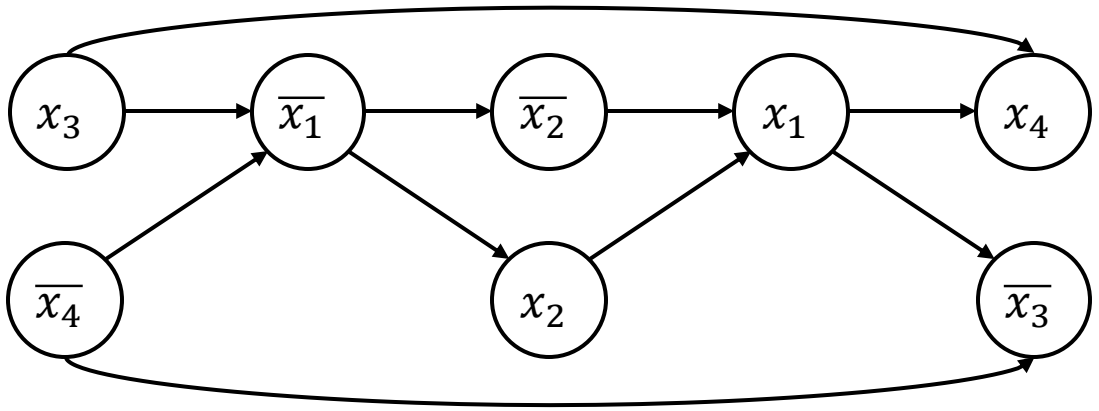


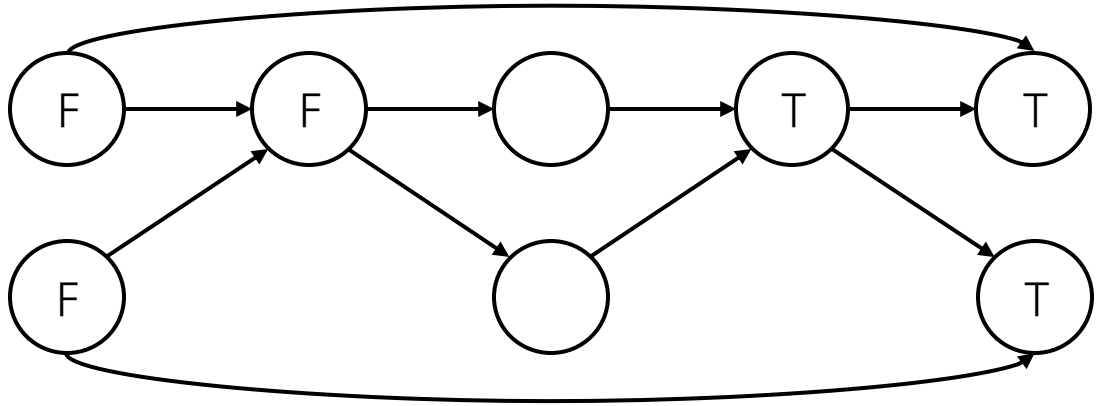
3.28. Here's an instance of 2SAT:

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

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 ith 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 $p_1, \ldots, p_n \in [0, 1]$, 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 ith 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: $x_1, \ldots, x_n; v$.
Question: Is it possible to make change for v using coins of denominations $x_1, \ldots, x_n$?

$dp[i]$: 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 …

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

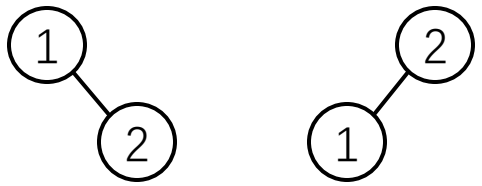
Complexity: $O(nv)$

6.20. Give an efficient algorithm for the following task.

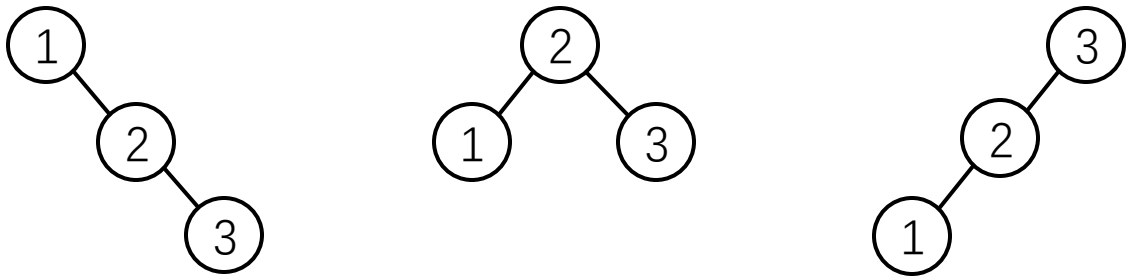
Input: n words (in sorted order); frequencies of these words: $p_1, p_2, \ldots, p_n$.

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: $p_1, p_2, \ldots, p_n$.

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].

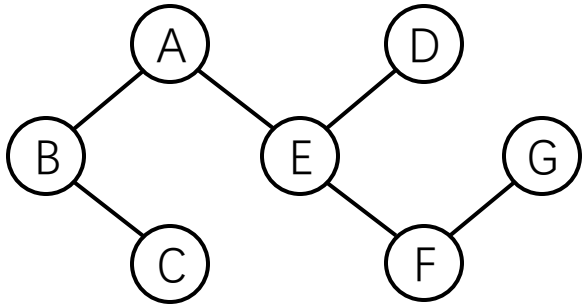
	1	2	3	4
1	0	$p_1 + p_2$	$\min(p_2 + p_3, 0 + 0, p_1 + p_2) + p_1 + p_2 + p_3$	
2	-	0	$p_2 + p_3$	$\min(p_3 + p_4, 0 + 0, p_2 + p_3) + p_2 + p_3 + p_4$
3	-	-	0	$p_3 + p_4$
4	-	-	-	0

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

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

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

Complexity: $O(|V|)$

6.22. Give an $O(nt)$ algorithm for the following task.
Input: A list of n positive integers $a_1, a_2, \ldots, a_n$; 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: $O(nt)$