

Bitmask_DP.cpp

```

1  /**
2   *   Author: devesh95
3   *
4   *   Topic: Bitmask DP Examples (Easy to Hard)
5   *
6   *   Description:
7   *   This file contains 20 different dynamic programming problems solved using bitmasking.
8   *   The examples are arranged roughly from easier (simple subset enumeration) to
9   *   harder (assignment, TSP, graph problems, etc.). Each example includes detailed
10  *   comments explaining the DP state, recurrence (DP equation), and input/output
    processing.
11  *
12  *   Compilation:
13  *       g++ -std=c++17 -O2 -Wall Bitmask_DP_Examples.cpp -o bitmask_dp
14  *
15  *   Execution:
16  *       ./bitmask_dp
17  */
18
19  #include <bits/stdc++.h>
20  using namespace std;
21
22  #define int long long
23  #define INF 1000000000 // Use a large value
24
25  // -----
26  // 1. Enumerate All Subsets
27  // -----
28  /*
29   Problem:
30   Given n, enumerate all 2^n subsets.
31
32   Explanation:
33   This example does not use a DP recurrence but uses bitmask enumeration.
34
35   (No DP equation)
36  */
37  void solve_enumerate_subsets() {
38      cout << "\n----- 1. Enumerate All Subsets ----- \n";
39      int n;
40      cout << "Enter n (number of elements): ";
41      cin >> n;
42      cout << "All subsets (each as bitmask):\n";
43      int total = 1 << n;
44      for (int mask = 0; mask < total; mask++) {
45          // Print only n bits
46          string bits = bitset<16>(mask).to_string().substr(16 - n);
47          cout << bits << "\n";
48      }
49  }
50
51  // -----

```

```

52 // 2. Sum of All Subsets
53 // -----
54 /*
55 Problem:
56 Given an array of n numbers, compute and display the sum of each subset.
57
58 Explanation:
59 We enumerate each subset and compute its sum.
60
61 (No DP equation)
62 */
63 void solve_sum_of_subsets() {
64     cout << "\n----- 2. Sum of All Subsets ----- \n";
65     int n;
66     cout << "Enter n (number of elements): ";
67     cin >> n;
68     vector<int> arr(n);
69     cout << "Enter " << n << " numbers:\n";
70     for (int i = 0; i < n; i++) cin >> arr[i];
71
72     int total = 1 << n;
73     for (int mask = 0; mask < total; mask++) {
74         int sum = 0;
75         cout << "Subset (mask " << bitset<16>(mask).to_string().substr(16 - n) << "): ";
76         for (int i = 0; i < n; i++) {
77             if(mask & (1 << i)){
78                 sum += arr[i];
79                 cout << arr[i] << " ";
80             }
81         }
82         cout << "=> Sum: " << sum << "\n";
83     }
84 }
85
86 // -----
87 // 3. Count Subsets with Given Sum
88 // -----
89 /*
90 Problem:
91 Given an array and a target sum S, count the number of subsets that sum to S.
92
93 DP Equation / Recurrence:
94 (Brute-force via bitmask enumeration; no memoized recurrence)
95
96 For each subset represented by mask, compute:
97 if (sum(mask) == S) then count++
98 */
99 void solve_count_subsets_with_sum() {
100     cout << "\n----- 3. Count Subsets with Given Sum ----- \n";
101     int n, S;
102     cout << "Enter n (number of elements) and target sum S: ";
103     cin >> n >> S;
104     vector<int> arr(n);
105     cout << "Enter " << n << " numbers:\n";

```

```

106     for (int i = 0; i < n; i++) cin >> arr[i];
107
108     int total = 1 << n;
109     int count = 0;
110     for (int mask = 0; mask < total; mask++) {
111         int sum = 0;
112         for (int i = 0; i < n; i++) {
113             if(mask & (1 << i))
114                 sum += arr[i];
115         }
116         if(sum == S) count++;
117     }
118     cout << "Number of subsets with sum " << S << ": " << count << "\n";
119 }
120
121 // -----
122 // 4. Maximum Sum Subset
123 // -----
124 /*
125 Problem:
126     Given an array of non-negative numbers, find the subset with the maximum sum.
127
128 Explanation:
129     (Trivially, the maximum sum is the sum of all elements if all are non-negative.)
130
131     (No DP equation; simple enumeration.)
132 */
133 void solve_max_sum_subset() {
134     cout << "\n----- 4. Maximum Sum Subset ----- \n";
135     int n;
136     cout << "Enter n (number of elements): ";
137     cin >> n;
138     vector<int> arr(n);
139     cout << "Enter " << n << " non-negative numbers:\n";
140     for (int i = 0; i < n; i++) cin >> arr[i];
141
142     int total = 1 << n;
143     int maxSum = 0;
144     for (int mask = 0; mask < total; mask++) {
145         int sum = 0;
146         for (int i = 0; i < n; i++) {
147             if(mask & (1 << i))
148                 sum += arr[i];
149         }
150         maxSum = max(maxSum, sum);
151     }
152     cout << "Maximum subset sum: " << maxSum << "\n";
153 }
154
155 // -----
156 // 5. Assignment Problem (Minimum Cost Matching)
157 // -----
158 /*
159 Problem:

```

```

160     Given an n x n cost matrix, assign each job to a worker so that the total cost is
        minimized.
161
162     DP Equation:
163     Let dp[mask] be the minimum cost when jobs represented by mask are assigned.
164     Transition:
165         dp[mask | (1<<j)] = min(dp[mask | (1<<j)], dp[mask] + cost[popcount(mask)][j])
166 */
167 void solve_assignment_problem() {
168     cout << "\n----- 5. Assignment Problem (Min Cost Matching) ----- \n";
169     int n;
170     cout << "Enter n (number of jobs/workers): ";
171     cin >> n;
172     vector<vector<int>> cost(n, vector<int>(n));
173     cout << "Enter the cost matrix (n x n):\n";
174     for (int i = 0; i < n; i++)
175         for (int j = 0; j < n; j++)
176             cin >> cost[i][j];
177
178     int N = 1 << n;
179     vector<int> dp(N, INT_MAX);
180     dp[0] = 0;
181     for (int mask = 0; mask < N; mask++) {
182         int i = __builtin_popcount(mask); // number of jobs assigned so far
183         for (int j = 0; j < n; j++) {
184             if (!(mask & (1 << j))) {
185                 dp[mask | (1 << j)] = min(dp[mask | (1 << j)], dp[mask] + cost[i][j]);
186             }
187         }
188     }
189     cout << "Minimum assignment cost: " << dp[N - 1] << "\n";
190 }
191
192 // -----
193 // 6. Traveling Salesman Problem (TSP)
194 // -----
195 /*
196     Problem:
197     Given n cities and a cost (distance) matrix, find the minimum cost to visit all cities
198     starting from 0 and returning to 0.
199
200     DP Equation:
201     Let dp[mask][i] be the minimum cost to reach city i with visited set = mask.
202     Transition:
203         dp[mask | (1<<j)][j] = min(dp[mask | (1<<j)][j], dp[mask][i] + dist[i][j])
204 */
205 void solve_tsp() {
206     cout << "\n----- 6. Traveling Salesman Problem (TSP) ----- \n";
207     int n;
208     cout << "Enter number of cities: ";
209     cin >> n;
210     vector<vector<int>> dist(n, vector<int>(n));
211     cout << "Enter the distance matrix:\n";
212     for (int i = 0; i < n; i++)

```

```

213     for (int j = 0; j < n; j++)
214         cin >> dist[i][j];
215
216     int N = 1 << n;
217     vector<vector<int>> dp(N, vector<int>(n, INT_MAX));
218     dp[1][0] = 0; // Starting at city 0 (mask 1 means only city0 visited)
219     for (int mask = 1; mask < N; mask++) {
220         for (int i = 0; i < n; i++) {
221             if (mask & (1 << i)) {
222                 for (int j = 0; j < n; j++) {
223                     if (!(mask & (1 << j)) && dist[i][j] < INT_MAX) {
224                         dp[mask | (1 << j)][j] = min(dp[mask | (1 << j)][j], dp[mask][i] +
dist[i][j]);
225                     }
226                 }
227             }
228         }
229     }
230     int ans = INT_MAX;
231     for (int i = 0; i < n; i++) {
232         ans = min(ans, dp[N - 1][i] + dist[i][0]);
233     }
234     cout << "Minimum TSP cost: " << ans << "\n";
235 }
236
237 // -----
238 // 7. Counting Hamiltonian Paths in a DAG
239 // -----
240 /*
241     Problem:
242     Given a directed acyclic graph (DAG) with n nodes, count the number of Hamiltonian
paths.
243
244     DP Equation:
245     Let dp[mask][i] be the number of ways to reach node i having visited nodes in mask.
246     Transition:
247         dp[mask | (1<<v)][v] += dp[mask][u], for each edge (u -> v) where v is not in mask.
248 */
249 void solve_count_hamiltonian_paths() {
250     cout << "\n----- 7. Counting Hamiltonian Paths in a DAG ----- \n";
251     int n, m;
252     cout << "Enter number of nodes and edges: ";
253     cin >> n >> m;
254     vector<vector<int>> graph(n);
255     cout << "Enter directed edges (u v) (0-indexed):\n";
256     for (int i = 0; i < m; i++){
257         int u, v;
258         cin >> u >> v;
259         graph[u].push_back(v);
260     }
261     int N = 1 << n;
262     vector<vector<int>> dp(N, vector<int>(n, 0));
263     for (int i = 0; i < n; i++)
264         dp[1 << i][i] = 1;

```

```

265     for (int mask = 0; mask < N; mask++){
266         for (int u = 0; u < n; u++){
267             if(mask & (1 << u)){
268                 for (int v : graph[u]){
269                     if(!(mask & (1 << v))){
270                         dp[mask | (1 << v)][v] += dp[mask][u];
271                     }
272                 }
273             }
274         }
275     }
276     int total = 0;
277     for (int i = 0; i < n; i++){
278         total += dp[N - 1][i];
279     }
280     cout << "Total Hamiltonian paths in the DAG: " << total << "\n";
281 }
282
283 // -----
284 // 8. Maximum Independent Set (Graph)
285 // -----
286 /*
287     Problem:
288     Given an undirected graph with n vertices (n small), find the size of the maximum
289     independent set.
290
291     Explanation:
292     Enumerate all subsets and check if the subset forms an independent set.
293
294     (No DP recurrence; brute-force bitmask enumeration)
295 */
296 void solve_max_independent_set() {
297     cout << "\n----- 8. Maximum Independent Set ----- \n";
298     int n, m;
299     cout << "Enter number of vertices and edges: ";
300     cin >> n >> m;
301     vector<vector<bool>> adj(n, vector<bool>(n, false));
302     cout << "Enter " << m << " edges (u v) (0-indexed): \n";
303     for (int i = 0; i < m; i++){
304         int u, v;
305         cin >> u >> v;
306         adj[u][v] = adj[v][u] = true;
307     }
308     int N = 1 << n;
309     int maxSize = 0;
310     for (int mask = 0; mask < N; mask++){
311         bool valid = true;
312         int count = 0;
313         for (int i = 0; i < n && valid; i++){
314             if(mask & (1 << i)){
315                 count++;
316                 for (int j = i+1; j < n; j++){
317                     if(mask & (1 << j)){
318                         if(adj[i][j]){

```

```

318         valid = false;
319         break;
320     }
321 }
322 }
323 }
324 }
325 if(valid)
326     maxSize = max(maxSize, count);
327 }
328 cout << "Size of maximum independent set: " << maxSize << "\n";
329 }
330
331 // -----
332 // 9. Maximum Clique (Graph)
333 // -----
334 /*
335     Problem:
336     Given an undirected graph with n vertices (n small), find the size of the maximum
337     clique.
338
339     Explanation:
340     Enumerate all subsets and check if they form a clique.
341
342     (No DP recurrence; brute-force enumeration)
343 */
344 void solve_max_clique() {
345     cout << "\n----- 9. Maximum Clique ----- \n";
346     int n, m;
347     cout << "Enter number of vertices and edges: ";
348     cin >> n >> m;
349     vector<vector<bool>> adj(n, vector<bool>(n, false));
350     // Mark self-loops for convenience.
351     for (int i = 0; i < n; i++) adj[i][i] = true;
352     cout << "Enter " << m << " edges (u v) (0-indexed):\n";
353     for (int i = 0; i < m; i++){
354         int u, v;
355         cin >> u >> v;
356         adj[u][v] = adj[v][u] = true;
357     }
358     int N = 1 << n;
359     int maxClique = 0;
360     for (int mask = 0; mask < N; mask++){
361         vector<int> nodes;
362         bool clique = true;
363         for (int i = 0; i < n; i++){
364             if(mask & (1 << i))
365                 nodes.push_back(i);
366         }
367         for (int i = 0; i < (int)nodes.size() && clique; i++){
368             for (int j = i+1; j < (int)nodes.size(); j++){
369                 if(!adj[nodes[i]][nodes[j]]){
370                     clique = false;
371                     break;

```

```

371         }
372     }
373 }
374 if(clique)
375     maxClique = max(maxClique, (int)nodes.size());
376 }
377 cout << "Size of maximum clique: " << maxClique << "\n";
378 }
379
380 // -----
381 // 10. Minimum Vertex Cover (Graph)
382 // -----
383 /*
384     Problem:
385     Given an undirected graph with n vertices (n small), find the size of the minimum
386     vertex cover.
387
388     Explanation:
389     A vertex cover is a set of vertices such that every edge is incident to at least one
390     vertex in the set.
391     (Brute-force enumeration via bitmask)
392
393     (No DP recurrence; relation: |MIS| + |MinVertexCover| = n)
394 */
395 void solve_min_vertex_cover() {
396     cout << "\n----- 10. Minimum Vertex Cover ----- \n";
397     int n, m;
398     cout << "Enter number of vertices and edges: ";
399     cin >> n >> m;
400     vector<vector<bool>> adj(n, vector<bool>(n, false));
401     cout << "Enter " << m << " edges (u v) (0-indexed): \n";
402     for (int i = 0; i < m; i++){
403         int u, v;
404         cin >> u >> v;
405         adj[u][v] = adj[v][u] = true;
406     }
407     int N = 1 << n;
408     int minCover = n;
409     for (int mask = 0; mask < N; mask++){
410         bool cover = true;
411         for (int u = 0; u < n && cover; u++){
412             for (int v = u+1; v < n && cover; v++){
413                 if(adj[u][v]){
414                     // At least one of u or v must be in the cover.
415                     if (!(mask & (1 << u)) && !(mask & (1 << v)))
416                         cover = false;
417                 }
418             }
419         }
420         if(cover){
421             int cnt = __builtin_popcount(mask);
422             minCover = min(minCover, cnt);
423         }
424     }
425 }

```



```

423     cout << "Minimum vertex cover size: " << minCover << "\n";
424 }
425
426 // -----
427 // 11. Set Cover Problem
428 // -----
429 /*
430 Problem:
431     Given a universe of m elements and n subsets (each represented as a bitmask),
432     find the minimum number of subsets required to cover the entire universe.
433
434 Explanation:
435     We enumerate over all subset selections and check if their union covers the universe.
436
437     (No DP recurrence; brute-force enumeration)
438 */
439 void solve_set_cover() {
440     cout << "\n----- 11. Set Cover Problem ----- \n";
441     int m, n;
442     cout << "Enter size of universe (m) and number of subsets (n): ";
443     cin >> m >> n;
444     vector<int> subsets(n);
445     cout << "Enter each subset as a bitmask (integer between 0 and " << ((1 << m) - 1) <<
446     "):\n";
447     for (int i = 0; i < n; i++){
448         cin >> subsets[i];
449     }
450     int full = (1 << m) - 1;
451     int N = 1 << n;
452     int ans = INT_MAX;
453     for (int mask = 0; mask < N; mask++){
454         int cover = 0, count = 0;
455         for (int i = 0; i < n; i++){
456             if(mask & (1 << i)){
457                 cover |= subsets[i];
458                 count++;
459             }
460         }
461         if(cover == full)
462             ans = min(ans, count);
463     }
464     if(ans == INT_MAX)
465         cout << "No cover found.\n";
466     else
467         cout << "Minimum number of subsets to cover the universe: " << ans << "\n";
468 }
469 // -----
470 // 12. Count Perfect Matchings in a Bipartite Graph
471 // -----
472 /*
473 Problem:
474     Given a bipartite graph with n workers and n jobs, count the number of perfect
475     matchings.

```

```

475
476 DP Equation:
477     Let dp[mask] be the number of ways to assign jobs corresponding to the bitmask.
478     Transition:
479         dp[mask | (1 << j)] += dp[mask] for each unassigned job j that can be matched.
480 */
481 void solve_count_perfect_matchings() {
482     cout << "\n----- 12. Count Perfect Matchings in a Bipartite Graph ----- \n";
483     int n;
484     cout << "Enter n (number of workers/jobs): ";
485     cin >> n;
486     vector<vector<bool>> adj(n, vector<bool>(n, false));
487     cout << "Enter the " << n << "x" << n << " bipartite adjacency matrix (0/1):\n";
488     for (int i = 0; i < n; i++){
489         for (int j = 0; j < n; j++){
490             int temp;
491             cin >> temp;
492             adj[i][j] = (temp == 1);
493         }
494     }
495     int N = 1 << n;
496     vector<int> dp(N, 0);
497     dp[0] = 1;
498     for (int mask = 0; mask < N; mask++){
499         int i = __builtin_popcount(mask);
500         for (int j = 0; j < n; j++){
501             if(!(mask & (1 << j)) && adj[i][j]){
502                 dp[mask | (1 << j)] += dp[mask];
503             }
504         }
505     }
506     cout << "Number of perfect matchings: " << dp[N - 1] << "\n";
507 }
508
509 // -----
510 // 13. Partition into Two Subsets with Minimum Difference
511 // -----
512 /*
513 Problem:
514     Given an array, partition it into two subsets so that the difference of their sums is
515     minimized.
516
517 Explanation:
518     Enumerate all subsets to determine one subset sum and use total sum to compute
519     difference.
520
521 (No DP recurrence; brute-force enumeration)
522 */
523 void solve_partition_min_difference() {
524     cout << "\n----- 13. Partition into Two Subsets (Min Difference) ----- \n";
525     int n;
526     cout << "Enter n (number of elements): ";
527     cin >> n;
528     vector<int> arr(n);

```

```

527     int total = 0;
528     cout << "Enter the elements:\n";
529     for (int i = 0; i < n; i++){
530         cin >> arr[i];
531         total += arr[i];
532     }
533     int N = 1 << n;
534     int best = INT_MAX;
535     for (int mask = 0; mask < N; mask++){
536         int sum = 0;
537         for (int i = 0; i < n; i++){
538             if(mask & (1 << i))
539                 sum += arr[i];
540         }
541         best = min(best, (int)abs(total - 2 * sum));
542     }
543     cout << "Minimum difference between two subsets: " << best << "\n";
544 }
545
546 // -----
547 // 14. Team Formation (Divide into Two Teams Minimizing Difference)
548 // -----
549 /*
550     Problem:
551     Given an even number of players with skill levels, split them into two teams (each with
n/2 players)
552     such that the difference in total skills is minimized.
553
554     Explanation:
555     Enumerate over all bitmasks with exactly n/2 bits set.
556
557     (No DP recurrence; brute-force enumeration)
558 */
559 void solve_team_formation() {
560     cout << "\n----- 14. Team Formation (Equal Teams) ----- \n";
561     int n;
562     cout << "Enter even n (number of players): ";
563     cin >> n;
564     if(n % 2 != 0) {
565         cout << "n must be even.\n";
566         return;
567     }
568     vector<int> skill(n);
569     cout << "Enter skill values:\n";
570     for (int i = 0; i < n; i++) cin >> skill[i];
571
572     int N = 1 << n;
573     int half = n / 2;
574     int best = INT_MAX;
575     for (int mask = 0; mask < N; mask++){
576         if(__builtin_popcount(mask) == half){
577             int sum1 = 0;
578             for (int i = 0; i < n; i++){
579                 if(mask & (1 << i))

```

```

580         sum1 += skill[i];
581     }
582     int sum2 = accumulate(skill.begin(), skill.end(), 0LL) - sum1;
583     best = min(best, (int)abs(sum1 - sum2));
584 }
585 }
586 cout << "Minimum skill difference between two teams: " << best << "\n";
587 }
588
589 // -----
590 // 15. Count Subset Sum Ways (Alternate Counting)
591 // -----
592 /*
593 Problem:
594 Count the number of ways to choose a subset from an array that sums to a given target
595 S.
596 Explanation:
597 This is similar to example 3 but explicitly counts the ways.
598
599 (No DP recurrence; brute-force bitmask enumeration)
600 */
601 void solve_count_subset_sum_ways() {
602     cout << "\n----- 15. Count Subset Sum Ways ----- \n";
603     int n, S;
604     cout << "Enter n (number of elements) and target sum S: ";
605     cin >> n >> S;
606     vector<int> arr(n);
607     cout << "Enter the elements:\n";
608     for (int i = 0; i < n; i++) cin >> arr[i];
609
610     int N = 1 << n;
611     int ways = 0;
612     for (int mask = 0; mask < N; mask++){
613         int sum = 0;
614         for (int i = 0; i < n; i++){
615             if(mask & (1 << i))
616                 sum += arr[i];
617         }
618         if(sum == S) ways++;
619     }
620     cout << "Number of ways to achieve sum " << S << ": " << ways << "\n";
621 }
622
623 // -----
624 // 16. Longest Hamiltonian Path (Maximizing Weight) [General Graph]
625 // -----
626 /*
627 Problem:
628 Given a weighted complete graph with n vertices, find the maximum total weight
629 path that visits every vertex exactly once (does not need to return to the start).
630
631 DP Equation:

```

```

632     Let dp[mask][v] be the maximum weight path ending at vertex v covering vertices in
mask.
633     Transition:
634         dp[mask | (1<<u)][u] = max(dp[mask | (1<<u)][u], dp[mask][v] + weight[v][u])
635 */
636 void solve_longest_path_bitmask() {
637     cout << "\n----- 16. Longest Hamiltonian Path ----- \n";
638     int n;
639     cout << "Enter number of vertices: ";
640     cin >> n;
641     vector<vector<int>> weight(n, vector<int>(n));
642     cout << "Enter the weight matrix:\n";
643     for (int i = 0; i < n; i++)
644         for (int j = 0; j < n; j++)
645             cin >> weight[i][j];
646
647     int N = 1 << n;
648     vector<vector<int>> dp(N, vector<int>(n, 0));
649     // Initialize: single vertex path has weight 0.
650     for (int i = 0; i < n; i++)
651         dp[1 << i][i] = 0;
652
653     int ans = 0;
654     for (int mask = 0; mask < N; mask++){
655         for (int u = 0; u < n; u++){
656             if(mask & (1 << u)){
657                 for (int v = 0; v < n; v++){
658                     if(!(mask & (1 << v))){
659                         dp[mask | (1 << v)][v] = max(dp[mask | (1 << v)][v], dp[mask][u] +
weight[u][v]);
660                         ans = max(ans, dp[mask | (1 << v)][v]);
661                     }
662                 }
663             }
664         }
665     }
666     cout << "Maximum weight of a Hamiltonian path: " << ans << "\n";
667 }
668
669 // -----
670 // 17. Count Independent Sets in a Graph
671 // -----
672 /*
673     Problem:
674     Given an undirected graph with n vertices (n small), count the total number of
independent sets.
675
676     Explanation:
677     Enumerate all subsets and count those that are independent.
678
679     (No DP recurrence; brute-force enumeration)
680 */
681 void solve_count_independent_sets() {
682     cout << "\n----- 17. Count Independent Sets ----- \n";

```

```

683     int n, m;
684     cout << "Enter number of vertices and edges: ";
685     cin >> n >> m;
686     vector<vector<bool>> adj(n, vector<bool>(n, false));
687     cout << "Enter " << m << " edges (u v) (0-indexed):\n";
688     for (int i = 0; i < m; i++){
689         int u, v;
690         cin >> u >> v;
691         adj[u][v] = adj[v][u] = true;
692     }
693     int N = 1 << n;
694     int count = 0;
695     for (int mask = 0; mask < N; mask++){
696         bool independent = true;
697         for (int i = 0; i < n && independent; i++){
698             if(mask & (1 << i)){
699                 for (int j = i+1; j < n; j++){
700                     if(mask & (1 << j)){
701                         if(adj[i][j]){
702                             independent = false;
703                             break;
704                         }
705                     }
706                 }
707             }
708         }
709         if(independent) count++;
710     }
711     cout << "Total number of independent sets: " << count << "\n";
712 }
713
714 // -----
715 // 18. Minimum Dominating Set (Graph)
716 // -----
717 /*
718     Problem:
719     A dominating set of a graph is a set of vertices such that every vertex is either in
720     the set
721     or adjacent to a vertex in the set. Find the size of the minimum dominating set.
722
723     Explanation:
724     Enumerate all subsets; for each, check if it is a dominating set.
725
726     (No DP recurrence; brute-force enumeration)
727 */
728 void solve_min_dominating_set() {
729     cout << "\n----- 18. Minimum Dominating Set -----\n";
730     int n, m;
731     cout << "Enter number of vertices and edges: ";
732     cin >> n >> m;
733     vector<vector<bool>> adj(n, vector<bool>(n, false));
734     // Each vertex dominates itself.
735     for (int i = 0; i < n; i++) adj[i][i] = true;
736     cout << "Enter " << m << " edges (u v) (0-indexed):\n";

```

```

736     for (int i = 0; i < m; i++){
737         int u, v;
738         cin >> u >> v;
739         adj[u][v] = adj[v][u] = true;
740     }
741     int N = 1 << n;
742     int ans = n;
743     for (int mask = 0; mask < N; mask++){
744         vector<bool> dominated(n, false);
745         for (int i = 0; i < n; i++){
746             if(mask & (1 << i)){
747                 for (int j = 0; j < n; j++){
748                     if(adj[i][j])
749                         dominated[j] = true;
750                 }
751             }
752         }
753         bool valid = true;
754         for (int i = 0; i < n; i++){
755             if(!dominated[i]) { valid = false; break; }
756         }
757         if(valid)
758             ans = min(ans, (int)__builtin_popcount(mask));
759     }
760     cout << "Minimum dominating set size: " << ans << "\n";
761 }
762
763 // -----
764 // 19. Task Ordering with Prerequisites
765 // -----
766 /*
767 Problem:
768     Given n tasks with prerequisites (each task i has a bitmask pre[i] that indicates
769     which tasks must be completed before i), count the number of valid orderings.
770
771 DP Equation:
772     Let dp[mask] be the number of valid orderings for tasks in mask.
773 Transition:
774     For each task i not in mask, if (mask & pre[i] == pre[i]), then:
775     dp[mask | (1 << i)] += dp[mask]
776 */
777 void solve_task_ordering() {
778     cout << "\n----- 19. Task Ordering with Prerequisites ----- \n";
779     int n;
780     cout << "Enter number of tasks: ";
781     cin >> n;
782     vector<int> pre(n, 0);
783     cout << "For each task i (0-indexed), enter a bitmask (as integer) representing
prerequisites:\n";
784     cout << "(For example, if task 2 requires tasks 0 and 1, enter 3 (binary 11))\n";
785     for (int i = 0; i < n; i++){
786         cout << "Prerequisites for task " << i << ": ";
787         cin >> pre[i];
788     }

```

```

789     int N = 1 << n;
790     vector<int> dp(N, 0);
791     dp[0] = 1;
792     for (int mask = 0; mask < N; mask++){
793         for (int i = 0; i < n; i++){
794             if(!(mask & (1 << i)) && ((mask & pre[i]) == pre[i])){
795                 dp[mask | (1 << i)] += dp[mask];
796             }
797         }
798     }
799     cout << "Total number of valid orderings: " << dp[N - 1] << "\n";
800 }
801
802 // -----
803 // 20. Maximum XOR Subset (Bitmask Enumeration)
804 // -----
805 /*
806 Problem:
807     Given an array of integers, find the maximum XOR value obtainable from any subset.
808
809 Explanation:
810     Enumerate all subsets and compute the XOR value.
811
812     (No DP recurrence; brute-force enumeration)
813 */
814 void solve_max_xor_subset() {
815     cout << "\n----- 20. Maximum XOR Subset ----- \n";
816     int n;
817     cout << "Enter n (number of elements): ";
818     cin >> n;
819     vector<int> arr(n);
820     cout << "Enter the elements:\n";
821     for (int i = 0; i < n; i++) cin >> arr[i];
822
823     int N = 1 << n;
824     int maxXor = 0;
825     for (int mask = 0; mask < N; mask++){
826         int curXor = 0;
827         for (int i = 0; i < n; i++){
828             if(mask & (1 << i))
829                 curXor ^= arr[i];
830         }
831         maxXor = max(maxXor, curXor);
832     }
833     cout << "Maximum XOR value from any subset: " << maxXor << "\n";
834 }
835
836 // -----
837 // Main Menu
838 // -----
839 int32_t main() {
840     ios_base::sync_with_stdio(false);
841     cin.tie(nullptr);
842

```



```
843 while(true) {
844     cout << "\n===== \n";
845     cout << "          Bitmask DP Examples - Menu\n";
846     cout << "===== \n";
847     cout << " 1. Enumerate All Subsets\n";
848     cout << " 2. Sum of All Subsets\n";
849     cout << " 3. Count Subsets with Given Sum\n";
850     cout << " 4. Maximum Sum Subset\n";
851     cout << " 5. Assignment Problem (Min Cost Matching)\n";
852     cout << " 6. Traveling Salesman Problem (TSP)\n";
853     cout << " 7. Counting Hamiltonian Paths in a DAG\n";
854     cout << " 8. Maximum Independent Set (Graph)\n";
855     cout << " 9. Maximum Clique (Graph)\n";
856     cout << "10. Minimum Vertex Cover (Graph)\n";
857     cout << "11. Set Cover Problem\n";
858     cout << "12. Count Perfect Matchings in Bipartite Graph\n";
859     cout << "13. Partition into Two Subsets (Min Difference)\n";
860     cout << "14. Team Formation (Equal Teams)\n";
861     cout << "15. Count Subset Sum Ways\n";
862     cout << "16. Longest Hamiltonian Path (Max Weight)\n";
863     cout << "17. Count Independent Sets (Graph)\n";
864     cout << "18. Minimum Dominating Set (Graph)\n";
865     cout << "19. Task Ordering with Prerequisites\n";
866     cout << "20. Maximum XOR Subset\n";
867     cout << "21. Run All Examples\n";
868     cout << "0. Exit\n";
869     cout << "Enter your choice: ";
870
871     int choice;
872     cin >> choice;
873     if(choice == 0) break;
874
875     switch(choice) {
876         case 1: solve_enumerate_subsets(); break;
877         case 2: solve_sum_of_subsets(); break;
878         case 3: solve_count_subsets_with_sum(); break;
879         case 4: solve_max_sum_subset(); break;
880         case 5: solve_assignment_problem(); break;
881         case 6: solve_tsp(); break;
882         case 7: solve_count_hamiltonian_paths(); break;
883         case 8: solve_max_independent_set(); break;
884         case 9: solve_max_clique(); break;
885         case 10: solve_min_vertex_cover(); break;
886         case 11: solve_set_cover(); break;
887         case 12: solve_count_perfect_matchings(); break;
888         case 13: solve_partition_min_difference(); break;
889         case 14: solve_team_formation(); break;
890         case 15: solve_count_subset_sum_ways(); break;
891         case 16: solve_longest_path_bitmask(); break;
892         case 17: solve_count_independent_sets(); break;
893         case 18: solve_min_dominating_set(); break;
894         case 19: solve_task_ordering(); break;
895         case 20: solve_max_xor_subset(); break;
896         case 21:
```

```
897         solve_enumerate_subsets();
898         solve_sum_of_subsets();
899         solve_count_subsets_with_sum();
900         solve_max_sum_subset();
901         solve_assignment_problem();
902         solve_tsp();
903         solve_count_hamiltonian_paths();
904         solve_max_independent_set();
905         solve_max_clique();
906         solve_min_vertex_cover();
907         solve_set_cover();
908         solve_count_perfect_matchings();
909         solve_partition_min_difference();
910         solve_team_formation();
911         solve_count_subset_sum_ways();
912         solve_longest_path_bitmask();
913         solve_count_independent_sets();
914         solve_min_dominating_set();
915         solve_task_ordering();
916         solve_max_xor_subset();
917         break;
918     default: cout << "Invalid choice.\n";
919     }
920 }
921
922 return 0;
923 }
924
```