Competitive Programming Roadmap

1. Introduction to C++

- Guide (W3schools) up to “C++ Conditions”

> Competitive programming doesn’t require a deep knowledge of C++.

Introductory problems

- abc240_a - Edge Checker
- abc220_a - Find Multiple

2. Loops and arrays

- Guide (W3schools) up to “C++ Arrays” (optional: “C++ Functions”)
- Time Complexity, with final quiz
- Sorting an array
- CPH (chapters 1, 2, 3 up to “Comparison operators”)

> The stars (⭐) represent an estimate of the difficulty of the problem.

Problems

- abc204_b - Nuts
- abc205_b - Permutation Check
- abc206_c - Swappable
- abc220_c - Long Sequence
- abc196_c - Doubled ⭐
- abc194_c - Squared Error ⭐

- abc193_c - Unexpressed ⭐
- abc258_c - Rotation ⭐⭐

> If you fail to solve a problem, read a solution sketch at the end of the document. If you are still stuck, read the editorial (explanation of the solution). On the other hand, if you have solved the problem it may still be useful to check out the solution sketch and the editorial, which often contain tips, alternative solutions or variants of the problem.

> Training resources:
>
> - CPH
> - USACO Guide (for now, General and Bronze section; later, up to Gold section)
> - AtCoder Problems (problems)
> - To check your skills, you can take part in the contests on Codeforces and AtCoder, getting a score. At least two contests are organized every week. You can already start participating!

3. STL data structures

- CPH (chapter 4)
- Sets and maps
- Other set operations

Easy problems

- abc229_c - Cheese
- cms_catalogo - Tieni aggiornato il catalogo
- cms_cestini - La camera dei cestini
- cses_1091 - Concert Tickets
- cses_1640 - Sum of Two Values ⭐
- abc235_c - The Kth Time Query ⭐⭐

Hard problems

> May be useful for hard problems: Sort + greedy

> Greedy = make the choice that looks better (e.g., the one with the most profit / least cost / etc.). Sometimes you can prove that a greedy strategy is optimal.

- abc241_d - Sequence Query ⭐
- cses_1161 - Stick Divisions ⭐⭐
- cses_1164 - Room Allocation ⭐⭐⭐
- 1474C - Array Destruction ⭐⭐⭐⭐

4. Binary search

- Binary search
- CPH (end of chapter 3)

> "Binary search on the answer": assuming you only need to determine if the answer is $\geq x$, does the problem become easier? If so, you can try binary searching x.

Problems

- 1612C - Chat Ban ⭐
- cses_2422 - Multiplication Table ⭐
- abc248_d - Range Count Query ⭐⭐
- cses_1085 - Array Division ⭐⭐
- 1670D - Very Suspicious ⭐⭐⭐
- 1610C - Keshi Is Throwing a Party ⭐⭐⭐⭐

5. Summary problems (easy)

- CPH (start of chapters 5, 6)
- Prefix sums

Problems (#1)

- abc237_b - Matrix Transposition
- abc217_c - Inverse of Permutation

- abc247_c - 1 2 1 3 1 2 1
- abc246_c - Coupon ★
- abc247_d - Cylinder ★★
- abc237_d - LR insertion ★★
- abc227_c - ABC conjecture ★★
- abc271_c - Manga ★★★

Problems (#2)

- abc252_b - Takahashi's Failure
- abc263_b - Ancestor
- abc272_b - Everyone is Friends
- abc262_c - Min Max Pair ★
- abc250_c - Adjacent Swaps ★★
- abc254_c - K Swap ★★
- abc267_c - Subarray Index * A ★★
- abc275_d - Yet Another Recursive Function ★★★

6. Summary problems (hard)

- CPH (start of chapters 5, 6)
- Prefix sums

Problems (#1)

- ois_brackets - Quantum Brackets ★★★
- abc216_d - Pair of Balls ★★★
- abc217_e - Sorting Queries ★★★
- ois_prankster - Practical Jokes ★★★
- abc236_d - Dance ★★★
- ois_butoaie - Save The Barrels ★★★★
- ois_wine - Wine Tasting Tour ★★★★
- abc272_e - Add and Mex ★★★★

Problems (#2)

- abc252_d - Distinct Trio ★★★
- arc145_a - AB Palindrome ★★★
- ois_reading - Reading Papers ★★★
- ois_andxor - Bitwise Party ★★★
- abc248_e - K-colinear Line ★★★★
- abc270_e - Apple Baskets on Circle ★★★★
- abc257_e - Addition and Multiplication 2 ★★★★
- abc227_d - Project Planning ★★★★★

7. Introduction to DP (#1)

- Introduction to DP
- CPH (chapter 7)

Easy problems

- cses_1634 - Minimizing Coins
- cses_1635 - Coin Combinations I
- cses_1638 - Grid Paths
- ois_police3 - Police Investigation 3
- cses_1636 - Coin Combinations II ★
- abc248_c - Dice Sum ★★

Hard problems

- abc267_d - Subsequence Index * A ★★★
- cses_1097 - Removal Game ★★★
- abc162_f - Select Half ★★★★
- 1303D - Fill The Bag ★★★★
- 1359D - Yet Another Yet Another Task ★★★★★
- abc227_f - Treasure Hunting ★★★★★★

8. Introduction to DP (#2)

Easy problems

- cses_1746 - Array Description
- cses_1744 - Rectangle Cutting
- cses_1639 - Edit Distance ★
- cses_1158 - Book Shop ★★
- cses_2413 - Counting Towers ★★
- 1673C - Palindrome Basis ★★

Hard problems

- abc270_d - Stones ★★★
- 1517D - Explorer Space ★★★
- abc266_e - Throwing the Die ★★★★
- preoii_yutaka - Sushi variegato ★★★★
  English statement:
  In how many ways can you partition an array into subarrays, if each subarray must contain distinct values?
  (You have to download `yutaka.cpp` in the attachments and implement the function.)
- preegoi_parkour - Sui tetti di Pisa ★★★★★
  English statement:
  You start from position 0 and you have to reach position N. For each i, you can jump from i to $i+k$ ($A_i \le k \le B_i$). The cost of a path is the maximum S_i over all reached positions ($0 \le i < N$). Find the minimum cost.
  (You have to download `parkour.cpp` in the attachments and implement the function.)
- 1765F - Chemistry Lab ★★★★★★

9. Introduction to computational number theory

- CPH (chapter 21)
- Fast exponentiation (up to “Large exponents modulo a number”)

- GCD computation (up to “Least common multiple”)
- Sieve of Eratosthenes (up to “Implementation”)
- Sieve with prime factors computation
- Factorization (up to “Trial division”)

Easy problems

- 1765M - Minimum LCM
- 1764B - Doremy’s Perfect Math Class
- abc276_d - Divide by 2 or 3 ★
- abc215_d - Coprime 2 ★★
- abc228_e - Integer Sequence Fair ★★
- abc280_d - Factorial and Multiple ★★★

Hard problems

- abc125_c - GCD on Blackboard ★★★
- arc118_c - Coprime Set ★★★
- preegoi_torta - Torta di Compleanno ★★★★
  English statement:
  You have to partition an array into $x \geq 2$ subarrays with the same sum. What’s the minimum possible x?
  (You have to download `torta.cpp` in the attachments and implement the function.)
- 1485D - Multiples and Power Differences ★★★★
- preoii_armadio - Evasione dall’armadio ★★★★
  (You have to download `armadio.cpp` in the attachments and implement the function.)
- arc122_c - Calculator ★★★★★

10. Introduction to graphs, DFS

- Connected components, trees (with final quiz)
- CPH (whole chapter 11, “depth-first search” and “connectivity check” in chapter 12)

Problems

- cses_1666 - Building Roads
- abc259_d - Circumferences ★★
- arc065_b - Connectivity ★★★
- ois_islands - Find the Treasure ★★★
- ois_slashes - Drawing Slashes ★★★
- ois_rainstorm - Flood Forecasting ★★★★

11. BFS

- Breadth First Search
- CPH (chapter 12)

Problems

- ois_water - Water Calculator ★
- abc272_d - Root M Leaper ★
- abc213_e - Stronger Takahashi ★★★
- ois_cannons - Circus Show ★★★
- arc084_b - Small Multiple ★★★★★
- oii_bus - Un lungo viaggio ★★★★★
  (You have to download `bus.cpp` in the attachments and implement the function.)

12. Introduction to DSU and Minimum Spanning Tree

- Disjoint Set Union
- Kruskal's Algorithm
- CPH (chapter 15, excluding "Prim's algorithm")

Problems

- cses_1666 - Building Roads (using DSU instead of DFS)

- cses_1676 - Road Construction
- cses_1675 - Road Reparation ★
- abc214_d - Sum of Maximum Weights ★★★★
- 1513D - GCD and MST ★★★★
- COCI 2017/6 - Sirni ★★★★★★

13. Dijkstra

- Dijkstra ($O(n + m \log n)$)
- CPH (chapter 13, only “Dijkstra’s algorithm”)

Problems

- cses_1671 - Shortest Routes I
- cses_1196 - Flight Routes ★★
- abc237_e - Skiing ★★★★
- abc191_e - Come Back Quickly ★★★★
- abc204_e - Rush Hour 2 ★★★★
- 1693C - Keshi in Search of AmShZ ★★★★★★

14. Final problems

Bonus problems

- abc241_e - Putting Candies ★★★
- preoii_triplets - Comune di Alleib ★★★
  English statement:
  You are given a tree. Let $\operatorname{dist}(x, y)$ be the length of the shortest path between x and y. Find the maximum $\operatorname{dist}(x, y) + \operatorname{dist}(y, z) + \operatorname{dist}(x, z)$.
  (You have to download `triplets.cpp` in the attachments and implement the function.)
- abc258_g - Triangle ★★★★
- ois_patrol - Police Patrol ★★★★

- abc197_f - Construct a Palindrome ★★★★
- agc054_b - Greedy Division ★★★★★
- preoii_sushi - Truffa al sushi ★★★★★★
  (You have to download `sushi.cpp` in the attachments and implement the function.)

Virtual contests

- abc184 - AtCoder Beginner Contest 184
- Codeforces Round #764 (Div. 3)

> During a contest, you should test your code locally (at least on sample cases) before submitting, to avoid penalty.

Bonus mashups

- Binary search mashup
- Number theory mashup
- DFS + BFS mashup (#1)
- DFS + BFS mashup (#2) (the problems are not sorted by difficulty)
- DP mashup
- Graphs mashup

Solutions

1. Introduction to C++

- Guide (W3schools) up to “C++ Conditions”

> Competitive programming doesn’t require a deep knowledge of C++.

Introductory problems

- abc240_a - Edge Checker

This problem checks if you can to write basic code in C++. In easier problems, the editorial usually contains a short implementation, which is worth reading after solving the problem.

- abc220_a - Find Multiple
  This problem can be solved using loops (covered in the next section), but there are solutions without loops.
  For sure, you can solve the problem “by hand” (given A, B, C, you know how to find a valid output if it exists). What operations do you perform “by hand”? Can you turn them into C++ code?

2. Loops and arrays

- Guide (W3schools) up to “C++ Arrays” (optional: “C++ Functions”)
- Time Complexity, with final quiz
- Sorting an array
- CPH (chapters 1, 2, 3 up to “Comparison operators”)

> The stars (⭐) represent an estimate of the difficulty of the problem.

Problems

- abc204_b - Nuts
  Iterate over i from 1 to n, adding up the number of nuts taken.

- abc205_b - Permutation Check
  For each integer from 1 to n, check if it is in the permutation. Alternatively, sort the array.

- abc206_c - Swappable
  Sort the array to count pairs of equal elements.

- abc220_c - Long Sequence
  You can’t construct the long array explicitly, but you can calculate the number of copies needed to get a sum $\geq x$. At that point you are only interested in the latest copy.

- abc196_c - Doubled ⭐
  10^{12} operations are too many, but valid strings are at most 10^6, and 10^6 operations are fine. To find valid strings, iterate over the first half and compute the corresponding number.

- abc194_c - Squared Error ⭐
  Use the fact that the A_i are small (so there are at most 401 distinct values). Instead of considering pairs of items, consider pairs of distinct values, after computing for each k ($-200 \leq k \leq 200$) how many $a_i = k$ there are.

> In many problems it is useful to "precalculate" something (in this case, how many $a_i = k$ there are), to perform subsequent operations faster (for example, "how many pairs (i, j) with $a_i = x$, $a_j = y$ are there?")

- abc193_c - Unexpressed ⭐
  The "expressed" numbers are few, and you can calculate them by iterating on b. Warning: you have to calculate how many distinct "expressed" numbers there are.

- abc258_c - Rotation ⭐⭐
  Explicitly rotating the string is too slow. Imagine "skipping" a query of type 1 just before processing a query of type 2: can you answer? Can you use the same trick to "skip" all the queries of type 1?

> If you fail to solve a problem, read a solution sketch here. If you are still stuck, read the editorial (explanation of the solution). On the other hand, if you have solved the problem it may still be useful to check out the solution sketch and the editorial, which often contain tips, alternative solutions or variants of the problem.

> Training resources:
>
> - CPH
> - USACO Guide (for now, General and Bronze section; later, up to Gold section)
> - AtCoder Problems (problems)
> - To check your skills, you can take part in the contests on Codeforces and AtCoder, getting a score. At least two contests are organized every week. You can already start participating!

3. STL data structures

- CPH (chapter 4)
- Sets and maps
- Other operations on sets

Easy problems

- abc229_c - Cheese
  Problem becomes easy after sorting cheese by A_i. The hard part is keeping the B_i matched to the corresponding A_i. It can be done using a `vector<array<int, 2>>`.

- cms_catalogo - Tieni aggiornato il catalogo
  Book IDs are large. What data structure can access large indices? A `map<long long, int>`.

- cms_cestini - La camera dei cestini
  In a `vector<int>` you can add or remove elements at the end of the vector. Each basket can be represented by a different vector: for example, you can use a `vector<vector<int>>`.

- cses_1091 - Concert Tickets
  The required operations are supported by a `set<int>`.

- cses_1640 - Sum of Two Values ⭐
  If you iterate over one of the two values, you can determine if the other exists with a `map<int, int>`. Warning: the positions of the two taken values must be distinct.

- abc235_c - The Kth Time Query ⭐⭐
  For each element, can you efficiently calculate all the positions where it is present? If yes, you can find the k-th easily. Before answering queries, precalculate the positions of each element in a `vector<vector<int>>`.

Hard problems

> May be useful for hard problems: Sort + greedy
> Greedy = make the choice that looks better (e.g., the one with the most profit / least cost / etc.). Sometimes you can prove that a greedy strategy is optimal.

- abc241_d - Sequence Query ⭐
  The required operations are supported by a `set<int>`. You have to use pointers (see

"other operations on sets" above).

- cses_1161 - Stick Divisions ★★
  Do the operations in reverse order. You can show that the sticks to be joined are always the two shorter ones. Simulate the process using a `set<int>` (or a `priority_queue<int>`, which supports fewer operations but is easier to use).
  Similar problem: abc252_f - Bread ★★★

> In some problems it is useful to consider the process in reverse (it may be more convenient to handle).

- cses_1164 - Room Allocation ★★★
  Every time a new guest arrives, use the free room with minimum index. To update the status of the rooms over time, sort all the events (departures and arrivals) in chronological order.

> In many problems it is useful to sort the "events".

- 1474C - Array Destruction ★★★★
  x decreases over time, but must always be greater than the maximum element in the array at that moment (otherwise you can't delete it). So, whenever you delete two elements, one of them must be the maximum (otherwise x becomes too small).
  n is small, so you can try all initial operations (they are $n-1$ because you have to take the maximum also at the beginning).

> When you think you can't try all the possibilities efficiently, look for "observations" that simplify the problem: in the previous case, the key idea to make the problem easier is to always take the maximum. Usually, such "observations" are relatively easy to prove (if you've found the observation, it's probably because you've already proved it), so make sure that the "observation" is correct before writing potentially wrong code.

4. Binary search

- Binary search
- CPH (end of chapter 3)

> "Binary search on the answer": assuming you only need to determine if the answer is $\geq x$, does the problem become easier? If so, you can try binary searching x.

Problems

- 1612C - Chat Ban ★
  With a formula, you can determine $f(k) =$ how many emojis are there in a triangle of "dimension" k. $f(k)$ is increasing, so you can calculate the minimum k such that $f(k) \geq x$ with a binary search.

- cses_2422 - Multiplication Table ★
  You can count the elements $\geq x$ in $O(n)$. Binary search only adds a $O(\log n)$ factor.

- abc248_d - Range Count Query ★★
  Have you already solved abc235_c - The Kth Time Query ★★ (section 3)? Add a binary search for each query. In this case you can avoid implementing the binary search and use the `lower_bound` and `upper_bound` functions.

- cses_1085 - Array Division ★★
  Suppose the maximum sum of a subarray is $\leq x$. Can you calculate the minimum number of subarrays in the division? You can iterate from left to right and make the subarrays as long as possible.

> Using binary search, we have "inverted the problem" from "what is the minimum possible maximum sum of a subarray if there are k subarrays?" to "what is the minimum number of subarrays if the sums are $\leq x$?". Binary search works very often in problems like "calculate the minimum possible maximum ..." or "calculate the maximum possible minimum ..." (minimax problems).

- 1670D - Very Suspicious ★★★
  How many regions can you create using x lines? Look for an optimal strategy. At that point it is easy to find a formula.

- 1610C - Keshi Is Throwing a Party ★★★★
  Do binary search on the number of people invited. Why is it good to know the number of invited people in advance? If you iterate over people in ascending order of wealth, you can easily determine how many richer people are invited only if you already know the final number of people.
  Now you can show that it's always optimal to invite someone if possible.

5. Summary problems (easy)

- CPH (start of chapters 5, 6)
- Prefix sums

Problems (#1)

- abc237_b - Matrix Transposition

  `b[i][j] = a[j][i]`

- abc217_c - Inverse of Permutation

  `q[p[i]] = i`

- abc247_c - 1 2 1 3 1 2 1

  Simulate the process, calculating S_i $(1 \leq i \leq N)$ in order.

- abc246_c - Coupon ★

  At most N coupons must be used partially.

- abc247_d - Cylinder ★★

  There are many elements of the same color. Use a `queue<array<long long, 2>>` containing pairs that store (color, number of consecutive elements of that color).

- abc237_d - LR insertion ★★

  Do the operations in reverse.

- abc227_c - ABC conjecture ★★

  Iterate on A, and for each A iterate on B. Can you calculate the complexity?

- abc271_c - Manga ★★★

  It is optimal to sell duplicate books, then books in descending order of volume. Alternatively, you can use a binary search.

Problems (#2)

- abc252_b - Takahashi's Failure

  Find the maximum tastiness. Is there food with maximum tastiness that Takahashi doesn't like?

- abc263_b - Ancestor

  Visit the ancestors of N until you find 1.

- abc272_b - Everyone is Friends
  For each pair of people, save if they met in a matrix. Can you update the matrix after every party?

- abc262_c - Min Max Pair ★
  There are two types of pairs ($a_i = i$, $a_i = j$). Count them separately.

- abc250_c - Adjacent Swaps ★★
  For each ball save its position in the array, and update the positions after each swap.

- abc254_c - K Swap ★★
  For each i, you can only sort elements in position i mod K.

- abc267_c - Subarray Index * A ★★
  Can you quickly update the score of a subarray if you move it one position to the right?
  Read more: Sliding Window

- abc275_d - Yet Another Recursive Function ★★★
  You don't need to calculate $f(x)$ for all x. To calculate $f(N)$, implement a recursive function that terminates immediately when called on an already calculated $f(x)$.
  Similar problems (they may require knowledge of DP, covered in section 7):
  agc044_a - Pay to Win ★★★★
  arc066_b - Xor Sum ★★★★★
  oii_compleanno - Emoji di compleanno ★★★★★
  (You have to download `compleanno.cpp` in the attachments and implement the function.)
  1670F - Jee, You See? ★★★★★★

6. Summary problems (hard)

- CPH (start of chapters 5, 6)
- Prefix sums

Problems (#1)

- ois_brackets - Quantum Brackets ★★★

Iterate over the brackets from left to right. It's always optimal to close a bracket if possible (in that case, remove the last pair of brackets). You can process the brackets using a `vector<int>`.

- abc216_d - Pair of Balls ★★★
  It is always optimal to remove a pair of balls if possible. Each cylinder is a `vector<int>`, so the configuration is a `vector<vector<int>>`.
  To efficiently find the pairs to remove, you need to know for each color how many times it appears at the top of a cylinder: these frequencies can be saved in an array and updated each time a ball is removed.
  When a frequency becomes equal to 2, that color can be removed. The colors to be removed can be saved in another `vector<int>`.
  Similar problem: oii_trendytrash - Pulizie d'autunno ★★★
  (You have to download `trendytrash.cpp` in the attachments and implement the function.)

> The implementation may seem tough: you can make it simpler by breaking the problem into smaller parts, each handled by a function. For example, you can write a function that removes a ball from a cylinder and performs all the required updates.

- abc217_e - Sorting Queries ★★★
  The sequence can be split into two parts (sorted, unsorted), represented by a `priority_queue<int>` and a `deque<int>`, respectively.

- ois_prankster - Practical Jokes ★★★
  Operations are applied in a subarray. To simplify operations consider, instead of the original array S, the array D defined as $D_i = S_i - S_{i-1}$. In this way, each operation modifies at most two elements of D. Now the problem is solved in a few lines.

> In some problems it is useful to consider the array of differences (as in the previous problem) or the array of prefix sums (the "inverse" of the array of differences, defined as $P_i = P_{i-1} + S_i$).

- abc236_d - Dance ★★★
  The only way to solve the problem is to try all possible pairings. They are actually less than $16!$ because some of them are counted multiple times. Consider this algorithm: "we take the first person not yet matched, and choose the other person in the pair". The configurations are therefore $15 \cdot 13 \cdot \dots \cdot 1$. A recursive function that finds all pairings this way is therefore efficient enough.

- ois_butoaie - Save The Barrels ★★★★
  Assuming $P > Q$, in one day you can use *ZeroBugs* in all barrels and *DiffBugs* (which removes only $P - Q$ insects) in K barrels. In this way, you have eliminated the "annoying" constraint of the $N - K$ barrels.
  Do binary search on the number of days x: In this way, you can remove xQ insects from all barrels, and now you only have to use *DiffBugs*.

- ois_wine - Wine Tasting Tour ★★★★
  Do binary search on the k-th sum (assuming it is $\leq x$). To count subarrays with sum $\leq x$, compute the prefix sums and for each left endpoint do binary search on the maximum right endpoint (or use Two Pointers).

- abc272_e - Add and Mex ★★★★
  Values not in $[1, n]$ are useless. During the process, a_i is in $[1, n]$ at most $n/i + 1$ times (because it always increases by i). In total, the useful values are therefore $O(n \log n)$ (by the harmonic series). For each operation, use a different `vector<int>` to save the useful values (so the configuration is a `vector<vector<int>>`). After finding the useful values, you can answer each query by sorting the corresponding vector.

Problems (#2)

- abc252_d - Distinct Trio ★★★
  Sort the elements (the answer does not change). Iterate on j and count the valid i, k.

- arc145_a - AB Palindrome ★★★
  Find configurations where the answer is "Yes" or "No", until you have handled all cases. For example, what if the string starts with `A` ?

- ois_reading - Reading Papers ★★★
  If you simulate the process from day 0 to day $L - 1$, it is not necessarily optimal to read the paper with more pages (there may be shorter but more "urgent" papers). Instead, simulating the process in reverse (from day $L - 1$ to day 0), valid papers never expire (but new papers may "appear"), and the greedy strategy works. You can save papers in a `priority_queue<int>` .

- ois_andxor - Bitwise Party ★★★
  If the AND is 0, all values must have one bit in common. Consider the elements with the bit i turned on (they are all different from 0). Then, the largest valid subset of

these elements includes them all (if the XOR is already different from 0) or all but one.
Then, for each i, it's enough to save and update the XOR.

- abc248_e - K-colinear Line ★★★★
  The distinct lines are $O(n^2)$, and each of them can be processed in $O(n)$.

- abc270_e - Apple Baskets on Circle ★★★★
  Do binary search on the number of full rounds. The incomplete round can be simulated in $O(n)$.

- abc257_e - Addition and Multiplication 2 ★★★★
  Find the number of digits. Now the goal is to maximize the digits in order from left to right. It is therefore optimal to take the maximum digit such that the remaining budget is sufficient to reach the maximum number of digits.

- abc227_d - Project Planning ★★★★★
  If the total number of employees is $\geq Kx$, and in every office there are $\leq x$ employees, you can complete x projects.
  Using this fact, you can determine the maximum number of projects with a binary search.

7. Introduction to DP (#1)

- Introduction to DP
- CPH (chapter 7)

Easy problems

- cses_1634 - Minimizing Coins
  `dp[i]` = minimum number of coins needed to get sum i. Can you calculate `dp[i]` if you know `dp[j]` ($j < i$)? Yes, if you fix the last used coin.

> A greedy method doesn't work (find a counterexample).

- cses_1635 - Coin Combinations I
  `dp[i]` = number of ways to get sum i. Basically, it's the same as cses_1634 - Minimizing Coins with the sum instead of the minimum.

- cses_1638 - Grid Paths
  `dp[i][j]` = number of legal paths from $(1, 1)$ to (i, j). If you fix the last move, you have already calculated the number of ways to complete the path.

- ois_police3 - Police Investigation 3
  `dp[i]` = minimum sum up to i, if you are forced to take T_i.

> Sometimes it is convenient to add additional constraints to the elements counted in the DP (in this case, "you are forced to take T_i"). In this way, the transitions are easier: it is immediate to get the distance between the last two stops (in the transition to `dp[i]` from `dp[j]` they are i, j), and verify that you have skipped at most one stop. Warning: the answer is not necessarily `dp[n]`.

- cses_1636 - Coin Combinations II ⭐
  `dp[i][j]` = number of ways to get sum j using the first i coins. To avoid using too much memory, you can avoid saving the "layer" with i (how?)
  Actually, the (iterative) solution to cses_1635 - Coin Combinations I works if you swap the two `for` loops.

> Note the difference between cses_1635 - Coin Combinations I and cses_1636 - Coin Combinations II ⭐: in the former, we iterate over the sums and for each sum over the coins (so the last coin taken can be any coin); in the second, we iterate over the coins and for each coin over the sums (thus, we are forced to take the coins in a fixed order).

> Memory optimization: if `dp[i][j]` only depends on `dp[i][*]` and `dp[i - 1][*]`, only save `dp[i][*]` and `dp[i - 1][*]` for each i, using an array `dp[2][m]`. In this problem, if you iterate on j the right order, you can just use a `dp[m]`. In general, it may happen that some DP values become useless, and therefore can be ignored and deleted, to optimize memory.

- abc248_c - Dice Sum ⭐⭐
  `dp[i][j]` = number of ways to get sum j with i dice.

Hard problems

- abc267_d - Subsequence Index * A ⭐⭐⭐
  `dp[i][j]` = maximum score considering only the elements up to i, and taking j elements in the subsequence.

> Suppose you take A_i in the subsequence. To calculate the contribution of A_i to the sum, you need to know its position in the subsequence (i.e. how many elements have already been taken before it). Also, the final subsequence must have length m. This suggests saving partial lengths of the subsequence (i.e., j).

- cses_1097 - Removal Game ★★★
  `dp[i][j]` = score of the first player, assuming that the game is played only in the interval $[i, j]$.
  Read more: Range DP

> In this case, the "states" of the DP coincide with the possible configurations of the game.

> Attention: the `dp[i][j]` must be calculated in the correct order. In most cases, it's fine to iterate by increasing i and increasing j. In this case, however, you cannot calculate `dp[1][n]` without knowing `dp[2][n]`. Note that each interval depends only on shorter intervals: this suggests iterating over the intervals in order of length.

- abc162_f - Select Half ★★★★
  Similar to abc267_d - Subsequence Index * A ★★★ (`dp[i][j]` = maximum score considering only the elements up to i, and taking j elements in the subsequence), but with bigger N.
  How many items can you take at most? Can you use the fact that you have to take a lot of elements (almost the maximum number possible) to your advantage? How many possible values of j are there for a fixed i?
  You can prove that $\lfloor i/2 \rfloor - 2 \leq j \leq \lfloor i/2 \rfloor + 2$. If you define $j := j - (\lfloor i/2 \rfloor - 2)$, the solution fits in the time limit.

> A trick to redefine j only once, before DP calculation: Arrays with negative indices in C++. This way, instead of writing every time `dp[i][j - ((i / 2) - 2)]`, you can just write `dp[i][j]`, even if j is large (and this does not reach the memory limit).
> The example in the blog maps $[-10^5, 10^5]$ to $[0, 2 \cdot 10^5]$, and similarly you can map $[\lfloor i/2 \rfloor - 2, \lfloor i/2 \rfloor + 2]$ to $[0, 4]$ by changing the function `int& dp(int i, int j)`.

- 1303D - Fill The Bag ★★★★
  If n is odd, the only way to fill the bag is to make a block of size 1. To do this, it is optimal to split the block of minimum size into two parts, until you have obtained a block of size 1. At this point, you can decrease n by 1.
  Now n is even (in any case). If necessary, create a block of size 2, splitting the one of

minimum size (but > 2) or joining blocks of size 1.
Can you generalize for any power of 2? Consider `dp[i]` = (current) number of blocks of size 2^i.
Similar problem: oii_riciclo - Rifiuti da riciclare ★★★★★
(You have to download `riciclo.cpp` in the attachments and implement the function.)

> To operate on the powers of 2 you may find useful Intro to Bitwise Operators.

- 1359D - Yet Another Yet Another Task ★★★★★
  In an array, you can compute the maximum sum of a subarray in $O(n)$: Maximum Subarray Sum.
  The second solution (Kadane's algorithm) can be modified to also take into account the maximum card taken. In particular, `dp[i][j]` = maximum sum of a subarray ending in i and which contains only elements $\leq j$, of which at least one is equal to j.
  The result is the maximum of the `dp[i][j] - j`.

> j can be negative: you can use Arrays with negative indices in C++.

> Consider this DP: " `dp[i][j]` = maximum sum of a subarray ending in i and which contains only elements $\leq j$.
> What guarantees that the maximum card taken in the optimal subarray in `dp[i][j]` has value exactly j? In general it's false, but it doesn't matter.
> In fact, suppose the subarray $[l, i]$ with sum = `dp[i][j]` has maximum $m < j$. Then its score is not `dp[i][j] - j`, but the answer can't be `dp[i][j] - j`, because `dp[i][m] = dp[i][j]` (because of the subarray $[l, i]$), and `dp[i][m] - m > dp[i][j] - j`.
> Thus, only subarrays whose maximum coincides with j can contribute to the answer.
> This "trick" seems very specific, but it is actually useful in various problems.

> Memory optimization (which is not required in this problem): in the last DP, the `dp[*][j]` are independent, so can just use a single `dp[n]` and recalculate it for each j.

- abc227_f - Treasure Hunting ★★★★★★
  If you try to determine the K maximums while building the paths, without additional information, you should save for each position the visited $A_{i,j}$ for all paths (because you have no way of determining which of these paths will be the optimal one), which is too slow.

This suggests looking for a criterion to quickly determine whether an $A_{i,j}$ is among the K maximums or not: in this way, among the paths up to (i, j) with the same number of maximums, the one with the minimum sum is always optimal.
Fix the k-th maximum (it's in the grid, so there are $O(HW)$ candidates). Now you can write `dp[i][j][l]` = minimum sum in a path from $(1, 1)$ to (i, j) where there are already exist l maximums. The total complexity is therefore $O(H^2W^2K)$.

> Determining the maximum cost is easier. Can you find it with $H, W \leq 400$?

8. Introduction to DP (#2)

Easy problems

- cses_1746 - Array Description
  When you choose x_i, you must guarantee $x_i - x_{i-1} \leq 1$. This suggests saving the last x_i. `dp[i][j]` = number of arrays of length i such that the condition holds in $[1, i]$ and $x_i = j$.

- cses_1744 - Rectangle Cutting
  `dp[i][j]` = minimum number of cuts to obtain a rectangle of size $i \times j$. The possible cuts (transitions) are $O(a + b)$, so the complexity is $O(ab(a + b))$.

- cses_1639 - Edit Distance ⭐
  `dp[i][j]` = edit distance between the first i characters of s and the first j characters of t. To find transitions, iterate over the type of move made on the last character of s and t.

> Transitions are more intuitive if the problem is formulated in the following equivalent way.
> Calculate the minimum cost to make s, t empty. The possible moves are:
>
> - remove the last character from s, from t or from both (cost 1);
> - if the last characters of s, t are the same, remove both (cost 0).
>
> Why are the two problems equivalent?

- cses_1158 - Book Shop ⭐⭐
  `dp[i][j]` = maximum number of pages, considering the first i books and taking a

subset with cost j.

> A "slow" DP is `dp[i][j][k]` = "is it possible to get k pages, considering the first i books and taking a subset with cost j?" Moving k from the indices of the states to the value of the states, we have avoided a dimension.
> Note that avoiding j produces another DP: `dp[i][k]` = minimum cost, considering the first i books and taking a subset with k pages.
> The difference is that k has 10^6 possible values, while j just has 10^5 (because it is $\leq x$), so it's better to avoid k.

- cses_2413 - Counting Towers ⭐⭐
  `dp[i][j]` = number of (incomplete) configurations of height i, such that in the row i there are j columns where a block ends (i.e., there are j gray segments in the bottom edge of the row i). The transitions to i from $i-1$ can be calculated by hand for each pair of j.

> Transitions between complete configurations are not easy to handle: in particular, blocks can have arbitrary height, so you would have to make $O(n)$ transitions for each state. In these cases, it may be convenient to also consider incomplete configurations to reduce the number of transitions.

- 1673C - Palindrome Basis ⭐⭐
  Basically it is cses_1636 - Coin Combinations II ⭐, using the fact that palindromes are few.

Hard problems

- abc270_d - Stones ⭐⭐⭐
  `dp[i]` = first player's score, assuming that the players start with a pile with i coins.

- 1517D - Explorer Space ⭐⭐⭐
  $(i,j) \to (i,j) = (i,j) \to (x,y) \to (i,j)$. Note that k must be even (why?), and let (x,y) be the cell visited after $k/2$ moves.
  The problem becomes: for each (i,j), what is the minimum score of a path of length $k/2$ which starts from (i,j)? To process transitions, fix the first move of the path.

- abc266_e - Throwing the Die ⭐⭐⭐⭐
  `dp[i]` = optimal expected value of the score, if you roll the die i times. Depending on the result of the first roll, you can decide whether to use some of the remaining

$i-1$ rounds or end the game.

- preoii_yutaka - Sushi variegato ★★★★
  `dp[i]` = number of ways to split the first i pieces of sushi.
  Let $j < i$ be the maximum j such that $v_j = v_i$. So, the last piece is $[k, i]$ with $j < k \leq i$.
  How to get $\sum_{k=j}^{i-1}$ `dp[k]` fast? Computing the prefix sums.

- preegoi_parkour - Sui tetti di Pisa ★★★★★
  The maximum height should be minimized, and this suggests (see section 4) a binary search on the maximum height. Now, `dp[i]` = "Can you arrive to position i?"

- 1765F - Chemistry Lab ★★★★★★
  Draw profits as a function of y (after fixing contracts) in a Cartesian plane. The value to maximize is the area under the graph. You can ignore the contracts strictly inside that area (they are useless).

9. Introduction to computational number theory

- CPH (chapter 21)
- Fast exponentiation (up to "Large exponents modulo a number")
- GCD computation (up to "Least common multiple")
- Sieve of Eratosthenes (up to "Implementation")
- Sieve with prime factors computation
- Factorization (up to "Trial division")

Easy problems

- 1765M - Minimum LCM
  Note that $\text{lcm}(a, b) = ab/\text{gcd}(a, b)$. So you have to minimize $(n - a)(a/\text{gcd}(a, n - a))$. Supposing without loss of generality $n - a \geq a$, you can prove easily that the factor on the right is 1, then $n - a$ is a multiple of a, then a is a divisor of n. You have to minimize $n - a$, so a is the maximum proper divisor of n.

> In this problem, the whole solution can be guessed without proving it. In more difficult problems, intermediate steps are usually required, and it is worth proving them to avoid going the wrong way.

> An alternative way to check the solution (or intermediate steps) is to write a program that solves the small cases and check that the output matches. In this problem, printing the output of small cases makes it easier to guess the solution. This approach is convenient if you feel that it takes little time to write a "slow" solution.

- 1764B - Doremy's Perfect Math Class
  You can write the GCD of the integers in the input (by simulating Euclid's algorithm), all its multiples (how?) and nothing else (why?).

- abc276_d - Divide by 2 or 3 ★
  You can only decrease exponents of 2 and 3 in the factorization of a_i. At the end, the exponents of a fixed p in each a_i must be equal (because the factorization is unique). You can calculate the number of moves by factoring a_i.

> Can you solve the problem with $N = 10^6$, $a_i \leq 10^{18}$, without factoring the a_i?

- abc215_d - Coprime 2 ★★
  Factorize the A_i. Now the problem becomes "for which i, for each p in the factorization of A_i, there are no other A_j multiples of p?" and can be solved by updating the frequencies of the prime factors.

> Since we have to factor many relatively small integers ($\leq 10^7$), a faster way is to precalculate the sieve of Eratosthenes, which allows you to find a prime that divides $x \leq 10^7$ in $O(1)$.

> Instead of factoring the A_i, you can iterate on p and on its multiples, improving the complexity (that becomes $O(m \log m)$, where m is the maximum A_i).

> Bonus problem: oii_accensione ★★★★
> (You have to download `accensione.cpp` in the attachments and implement the function.)
> Probably the hardest part is optimizing from $N \leq 10^5$ to $N \leq 10^6$.

- abc228_e - Integer Sequence Fair ★★
  Can you implement fast exponentiation? Pay attention to edge cases: for example, N multiple of the module.

- abc280_d - Factorial and Multiple ★★★
  Assuming $N \leq 10^6$, can you solve the problem? What if $N > 10^6$?

Hard problems

- abc125_c - GCD on Blackboard ★★★
  You can assume that you delete any integer from the blackboard (why?), and the problem becomes "for each i, compute the GCD after deleting a_i". How? By calculating prefix and suffix GCDs.

- arc118_c - Coprime Set ★★★
  With a few tries you can get that
  – $\{6, 10, 15\}$ satisfies the conditions;
  – it makes sense to try to add one item at a time. In particular, starting from $\{6, 10, 15\}$ (which already has GCD = 1), as long as the new elements are not coprime to any of the elements already present.
  In particular, adding a multiple of an element that already exists works.
  One possible solution is to print multiples of $6, 10, 15$.

> Solve the problem with $N \leq 2900$.

- preegoi_torta - Torta di Compleanno ★★★★
  The number of slices must be a divisor of the total tastiness of the cake. The number of slices must be minimized: in particular, if it is not prime, for each proper divisor k, you can “merge” blocks of k adjacent slices, getting a better division.
  How to quickly determine which divisors of a “large” number are prime? For any divisor, just check if it is a multiple of a prime divisor already found.

- 1485D - Multiples and Power Differences ★★★★
  If x, y are two random integers, there is usually no z such that $|x - z|$ and $|y - z|$ are both fourth powers. This suggests putting equal numbers into pairs of cells at distance 2. In particular, by using a checkerboard pattern, you can put $\text{lcm}(1, \dots, 16) = 720720$ in all the “even” cells. Now what to put in the “odd” cells?

- preoii_armadio - Evasione dall’armadio ★★★★
  By manipulating the equation and using the definition of φ, you can get that the result is $\sum_{d|N, d\geq 3} \varphi(d-1)$. The $\varphi(i)$ can be precalculated with a method similar to the sieve of Eratosthenes.
  Read more: Euler’s totient function

- arc122_c - Calculator ★★★★★

The operations performed in reverse look like a non-optimized Euclid's algorithm. Even if you use `x -= y` instead of `x %= y` in Euclid's algorithm, if x, y are random the number of operations required is usually quite small. In particular, if $x = N$ and $y < x$ is a random number coprime with N, about one of three times you need ≤ 130 operations to get 1. So let's try random y, until we find one that satisfies the limit of operations.

10. Introduction to graphs, DFS

- Connected components, trees (with final quiz)
- CPH (whole chapter 11, "depth-first search" and "connectivity check" in chapter 12)

Problems

- cses_1666 - Building Roads
  Implement DFS, finding a representative for each connected component.

- abc259_d - Circumferences ★★
  Construct a graph where each circumference is a node, and two nodes are connected by an edge if the two corresponding circumferences intersect (how to check it?).

> In some problems where the input is not a graph, the solution uses graph algorithms (so you have to "find" and construct a suitable graph).

- arc065_b - Connectivity ★★★
  The DFS finds, for each node, the connected component it belongs to. We can give each component an ID (for example, its representative). To verify that two nodes are connected by both roads and tracks, you can check that the pairs (ID of car component, ID of train component) are equal. Now just count the number of equal pairs.

- ois_islands - Find the Treasure ★★★
  DFS on a grid. To simplify the implementation: Flood Fill

- ois_slashes - Drawing Slashes ★★★
  DFS on a grid. How do edges look like?

- ois_rainstorm - Flood Forecasting ★★★★

Do binary search on the answer, checking if the graph is connected with a DFS.

11. BFS

- Breadth First Search
- CPH (chapter 12)

Problems

- ois_water - Water Calculator ★
  BFS where each integer is a node. Build only the necessary nodes and terminate the BFS as soon as you arrive to node 1.

- abc272_d - Root M Leaper ★
  BFS on a grid.

- abc213_e - Stronger Takahashi ★★★
  The BFS calculates for each box how many moves are needed to reach it. In one move, which squares can you unlock?
  To manage edges of weight 0, you can implement a 0-1 BFS.

- ois_cannons - Circus Show ★★★
  Moving a cannon creates an edge from the initial to the final destination. The edges are too many, but in fact one edge of weight 1 for each pair of adjacent positions is enough.

> If there are too many edges, it is sometimes possible to “compress” the graph by removing unnecessary edges.

- arc084_b - Small Multiple ★★★★★
  The node x stores the “distance” (i.e. the minimum sum of the digits) of the integers $\equiv x \pmod K$. The graph can be constructed by adding one digit at a time: the edges represent the operations "add 1" (cost 1) and "multiply by 10" (cost 0).

- oii_bus - Un lungo viaggio ★★★★★
  There are several solutions. Potentially useful observations:
  - pairs (bus, bus stop) are few and could be used as nodes;
  - given a subset S of nodes, to create a directed edge of weight 0 for each pair

(S_i, S_j) with $i < j$, it's enough to create edges (S_i, S_{i+1});
- given a subset S of nodes, to create an undirected edge of weight 1 for each pair (S_i, S_j), you can create a new node S^*, connected to each node in S with two directed edges, one of weight 0 and one of weight 1.

12. Introduction to DSU and Minimum Spanning Tree

- Disjoint Set Union
- Kruskal's Algorithm
- CPH (chapter 15, excluding "Prim's algorithm")

Problems

- cses_1666 - Building Roads (using DSU instead of DFS)
  Insert edges using a DSU. You can choose the root of each tree as the representative of the corresponding connected component.

- cses_1676 - Road Construction
  Implement the DSU by saving the size for each component.

> If the DSU is implemented with "union by size", for each component you can save not only the size, but also all nodes of the component (in a vector).

- cses_1675 - Road Reparation ★
  Implement the minimum spanning tree.

- abc214_d - Sum of Maximum Weights ★★★★
  For each edge, consider its contribution to the answer. The problem then becomes "for each edge, in how many paths is it the maximum?".
  If you insert the edges in increasing order of weight, each edge connects two connected components, and the paths to be counted are those with a node in each of the two components. It is therefore sufficient to maintain a DSU with the dimensions of the components and add, for each edge, $w_i \cdot x \cdot y$, where x, y are the sizes of the components to be merged.

- 1513D - GCD and MST ★★★★
  To solve the problem you should "understand" the structure of the graph first. In

particular, if a subarray $[l, r]$ satisfies the conditions and its minimum is in position m, the subarrays $[i, m]$ and $[m, j]$ ($l \leq i \leq m, m \leq j \leq r$) also satisfy the conditions.
Therefore we get that, using Kruskal, at any moment the connected components correspond to subarrays, and the edges described above are enough to construct the minimum spanning tree. So just iterate over the a_i in ascending order and expand the corresponding subarray as much as possible to the left and right, possibly merging it with already existing subarrays.

- COCI 2017/6 - Sirni ★★★★★★
  There are too many edges, so it may make sense to remove unnecessary edges. After a few tries, you can get that the edge (a, b) is useless if there is $c < a$ such that $\lfloor c/b \rfloor = \lfloor a/b \rfloor$, because its weight is the sum of weights of (a, c) and (c, b).
  If you fix b and $\lfloor a/b \rfloor$, there are few edges (by the harmonic series). Then, how to find the minimum a?

13. Dijkstra

- Dijkstra ($O(n + m \log n)$)
- CPH (chapter 13, only "Dijkstra's algorithm")

Problems

- cses_1671 - Shortest Routes I
  Implement Dijkstra.

> Dijkstra calculates the distance from a node to all other nodes. If you invert the graph, you can calculate the distance from all nodes to a fixed node. However, there are no practical ways to compute the distance between each pair of nodes with better complexity than Dijkstra used n times, but there is a shorter way in $O(n^3)$: Floyd-Warshall.

- cses_1196 - Flight Routes ★★
  Modify Dijkstra. For each node, save the k minimum distances in a priority queue, and once a new distance is found put it in the priority queue to continue the path. Why can you ignore the partial distances that are not in the k minimums?

- abc237_e - Skiing ★★★★
  The problem asks for the “maximum happiness” (F_i), but the goal is to calculate “the minimum distance” (D_i), for some definition of distance. Furthermore, there must be no edges with negative weights.
  Both goals can be achieved by defining $D_i = H_i - F_i$ and changing the weights of the edges accordingly.

> Bellman-Ford can be used to handle edges with negative weights, but it is slow. Edges with negative weights usually have to be avoided by changing the definition of distance.

- abc191_e - Come Back Quickly ★★★★
  $i \to i = i \to j \to i$. Calculate the distance between each pair of nodes, and for each i iterate on j to find the path $i \to j \to i$ with minimum length.

- abc204_e - Rush Hour 2 ★★★★
  You can use Dijkstra, but calculating the optimal starting time every time you process an edge. Can you find a closed formula? Does ternary search work?

- 1693C - Keshi in Search of AmShZ ★★★★★★
  For each node i and for each $k \leq \deg(i)$, you want to calculate the distance (= minimum time needed to get to n) if you keep k outgoing edges open (those arriving at the nodes at the minimum distance) and block the remaining $\deg(i) - k$ edges.
  Note that the nodes at minimum distance are the first to be “finalized” in Dijkstra: therefore, once a node is finalized, you can update the adjacent nodes in the inverted graph, increasing the corresponding k. Like in “standard” Dijkstra, the node at minimum distance can be finalized, even if the distance has not yet been computed for all k (because increasing k requires to pass through an unfinalized node, getting a greater distance).

14. Final problems

Bonus problems

- abc241_e - Putting Candies ★★★
- preoii_triplets - Comune di Alleib ★★★
- abc258_g - Triangle ★★★★

- ois_patrol - Police Patrol ★★★★
- abc197_f - Construct a Palindrome ★★★★
- agc054_b - Greedy Division ★★★★★
- preoii_sushi - Truffa al sushi ★★★★★★

Virtual contests

- abc184 - AtCoder Beginner Contest 184
- Codeforces Round #764 (Div. 3)

> During a contest, you should test your code locally (at least on sample cases) before submitting, to avoid penalty.

Bonus mashups

- Binary search mashup
- Number theory mashup
- DFS + BFS mashup (#1)
- DFS + BFS mashup (#2) (the problems are not sorted by difficulty)
- DP mashup
- Graphs mashup