A Tour of Programming for Problem Solving

English Translation (CC BY 4.0) of the original text by Tomoyuki Kaneko (kaneko@acm.org)

This text is an automated translation of an original document authored by tkaneko.

The translation was produced using the Gemini 2.0 translation engine, followed by manual revisions to the LaTeX code to rectify incompatibilities.

This translation should be considered a preliminary experimental output.

Specifically, please note that a figure is absent from the Interpolation chapter, and some text sections may be duplicated or unnecessary.

While the material may be of informational value, the translation is offered in a rough, unedited state and warrants substantial revision.

Contents

Chapter 1

Introduction

1.1 Structure of the Material

Each chapter is designed to take 90-105 minutes of exercise time[1]. The ”Examples” and easy ”Exercises” with hints are mainly intended for beginners, and many of them can be solved in 5-10 minutes once understood. However, if you are tackling them for the first time, we recommend that you estimate about 5 times longer. For experienced users, multiple exercises with different levels of difficulty are provided. The time required varies depending on the level of proficiency, but as a guideline, when a problem name has one $\star$ mark, the difficulty increases by about 5 times (for example, measured by the time required to create an answer). Also, some problems may assume knowledge from later chapters. Therefore, instead of solving all the problems in each chapter before moving on to the next, we recommend that you move on to the next chapter once you have solved the easy problems without a mark. Once you have gone through the examples and seen what topics are covered, you will likely find that you can solve more problems in the second round. Furthermore, problems with two or more $\star$ marks may have some relation to the content of the chapter, but may not be directly related to the solution method. This is because determining which strategy is effective is one of the interesting aspects of learning problem-solving.

Legend Links within the material are shown in dark green (e.g., Chapter 1, Reference [3]). Also, external links are shown in blue (e.g., `http://www.graco.c.u-tokyo.ac.jp/icpc-challenge/`).

Supplementary Note for Students (Planning to) Take Practical Programming in the First Semester of the College of Arts and Sciences The seminars that will be held from now on do *not* assume that you have studied this material in advance. In other words, materials for beginners will continue to be provided alongside exercises for experienced users. The themes covered will overlap with this material in some parts and not in others.

[1] This was the initial intention, but due to organizational reasons, some chapters may not have the appropriate amount of content now.

1.2 Programming Languages

The following languages are considered for learning:

- **C++** (Recommended: This is the main target language)
- Java
- **Python3** (Recommended: However, some problems may not be solvable due to time limits)
- Ruby (Some problems may not be solvable due to time limits)

Learning with the following languages is not recommended:

- C (Reason: Due to the limited standard library. For example, it lacks associative array functionality.)
  C language users should use C++. The functionality required for the exercises in this material is only a small part of the whole C++, so you can borrow only that part and write the rest as if it were C. In other words, it should be less difficult than generally relearning C++.
- Python2 (Reason: There are various pitfalls such as variable scope and character encoding)

1.3 Textbooks and References

It is assumed that the reader of this material is able to write short code involving loops and conditional branches as intended, and has also learned about recursion. Therefore, if you are truly encountering programming for the first time, it is recommended that you first learn from another book. If you already have a purchased book, that is sufficient, but if you are purchasing a new one, ”Introduction to C/C++ Programming Starting with Online Judge” [1] has a good connection with this material in that it uses AOJ.

In the text, we may refer to Reference[3] as ”Algorithm and Data Structures for Competitive Programming” and Reference$^{\text{Strategy}}$[2] as ”Programming Contest Challenge Book, Second Edition” (it goes without saying that these are masterpieces in this field). Also, for those who have plenty of study time (and budget), we recommend taking the time to read up to Chapter 6 of ”Algorithm Design” [4]. Although it has many pages, it is written carefully, so it seems suitable for beginners among similar books (however, the author read the English version, so the evaluation of the Japanese version is an estimate). If you want to learn more deeply, ”Introduction to Algorithms” [5] is also worth spending time studying.

1.4 Online Judge Systems

The problems in this material are taken from the following online judges. When creating the material, the person in charge checked the terms of use of each online judge to the extent possible, and judged that there would be no problem in referring to each problem in this material. However, please contact us if you notice anything.

- Aizu Online Judge () `http://judge.u-aizu.ac.jp`
- Peking University Judge Online for ACM/ICPC () `http://poj.org`
- Codeforces `http://www.codeforces.com/`
- szkopul `https://szkopul.edu.pl` (formerly `http://main.edu.pl/en`)

When describing the sources, those from the domestic (`https://icpc.iisf.or.jp/`) and the ACM-ICPC Alumni Association (JAG; `http://acm-icpc.aitea.net/`) are simplified as much as possible within the range that they can be distinguished. For example, 国内予選 refers to the Japanese ACM-ICPC, and the mock domestic preliminary refers to the regular practice session organized by the ACM-ICPC Alumni Association.

1.5 Aizu Online Judge (AOJ)

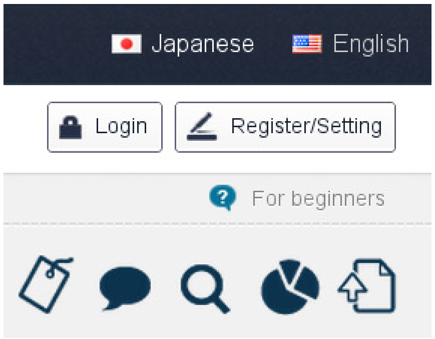
Creating an Account

If you are a first-time user, you will need to create an account on AOJ. All systems can be used free of charge. Please remember that each online judge is provided as a courtesy by its operators, so please use it in a way that does not cause any trouble. In particular, *do not forget your password*.

Creating an AOJ Account (First Time Only) Create an account from the Register/Setting link in the upper right corner of the page. Remember your User ID and Password (save it in your browser or in an encrypted file). Note that this communication is not https. For Affiliation, enter something like the University of Tokyo. You do not need to fill in E-mail or URL.

Here, you can choose whether to make your submitted programs public or not. If you make them public (select "public"), it may be convenient when asking for help with errors in your program. On the other hand, if you are trying to "test the behavior of other people's code snippets," copyright issues may arise, so it is better to keep them private (select "private").

Submitting to AOJ (Every Time) After logging in, with the problem statement displayed, click the icon with an upward arrow in a rectangle to display the form.



If the “Status” of the row corresponding to your submission (see “Author”) is “Accepted”, then your answer is correct.

When the Answer is Not Correct There can be various reasons, so first, read the explanations to see which of the judge’s responses applies and whether you have misunderstood how to use the system. There are materials such as Terms of use (`https://onlinejudge.u-aizu.ac.jp/#/term_of_use`), Judge’s replies (`https://onlinejudge.u-aizu.ac.jp/#/judges_replies`), and a tutorial (`http://judge.u-aizu.ac.jp/onlinejudge/AOJ_tutorial.pdf`).

In general, **it is common for programs not to work as intended, even for experienced programmers!**. Even if the program you have built does not work, it does not mean that everything is wrong. In many cases, it can be solved with a few small changes. Therefore, it is effective to check the operation of each part to see up to which point it is working correctly. Programming on a computer has the advantage that **you can copy and undo**, so (unlike cooking, where you might be disappointed by wasting ingredients), do not be afraid to try various things.

There is also some know-how on how to recover from difficult situations (Appendix A), so it will be useful to gradually acquire it. On the other hand, at an early stage of experience, we recommend that you *do not spend more than 15 minutes worrying*. Spending time worrying without a clue is not only painful, but also not very effective for learning at an early stage. It is better to rely on an instructor, senior, or friend, or to put it on hold and work on other problems to gain experience. When consulting, it will be quicker to solve the problem if you verbalize it by specifying ”it should work like this (the reason is this), but it actually works like that”. Note that the amount of time you can spend worrying meaningfully will increase as you become more proficient, such as 2 hours or 2 days.

1.6 Input/Output and Problem Correspondence

Let’s try submitting a solution to a problem on AOJ.

Example **Rectangle** (AOJ)

Create a program to calculate the area and perimeter of a rectangle with a width of a cm and a height of b cm. `http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ITP1_1_C&lang=jp`

Follow these steps:

1. Understand the problem.
2. Consider the calculation procedure (in this case, understand that you need to calculate $a * b$ and $2 * (a + b)$).
3. Write the program. Edit it with your preferred (Emacs, mi, etc.) and save it to a file.
4. Check the operation on your computer (required).
5. Submit to AOJ and confirm.

Example Solutions : Below are example solutions in each language and how to check the operation for the case of `a=3` and `b=5`.

Python3

```python
a,b = map(int, input().split())
area = a*b
perimeter = 2*(a+b)
print(area, perimeter)
```

`input()` reads a line, `split()` splits it by spaces[2], and `map(int, ...)` converts each split element to an integer.

After saving the above content as `rectangle.py`, execute it on the . `$` is an abbreviation for the prompt and does not need to be entered by yourself (that is, type after `python3`)[3]. Also, `#` and the part to its right are supplementary explanations and do not need to be entered. Press the Enter key at the end of each line of input. Below, italics indicate input from the keyboard.

```
$ python3 rectangle.py
3 5 # Input from the keyboard
15 16 # Program output display
```

[2] At this time, the trailing newline character and (if any) leading or consecutive spaces are also removed.

[3] See HWB15.2 for details `https://hwb.ecc.u-tokyo.ac.jp/wp/information-2/cui/terminal/`

C++

```cpp
#include <iostream>
using namespace std;
int main() {
  int a,b;
  cin >> a >> b;
  int area = a*b;
  int perimeter = 2*(a+b);
  cout << area << '␣' << perimeter << "\n";
}
```

Save the above program as `rectangle.cc`, etc. Then compile and execute it.
An example of operation on the ”Terminal” (in the case of MacOSX) is as follows:

```
$ g++ -std=c++11 -Wall -fsanitize=undefined rectangle.cc # Compile
$ ./a.out # Execute
3 5 # Input from the keyboard
15 16 # Program output display
```

# and everything to its right are comments and do not need to be entered. -std=c++11 is a that enables the C++11 standard. -Wall is an option that enables various warnings, and it is desirable to resolve any warning messages that appear. In particular, *if there is a question about the program's operation*, *resolve any prominent warnings before asking a question*. If you get a message that you don't know how to read, consult with someone. -fsanitize=undefined enables a mechanism to catch runtime errors. It is desirable to keep it on during operation checks. See Chapter A.2.2 for details.

C Language Prohibited

When reading this material, C language is insufficient, and it is necessary to use some of the functions of C++. We will introduce the necessary parts little by little, so get used to `iostream`, `cin`, `cout` etc. at this point.

Prohibition of Programming on the Browser

You may be able to code simple problems on the browser, but this is not the case for the complex problems we will be dealing with in the future. Prepare an environment where you can compile and run on your PC with various data at this point.

C++

```
#include <cstdio>
int main() {
  int a,b;
  scanf("%d␣%d", &a, &b);
  int area = a*b;
  int perimeter = 2*(a+b);
  printf("%d␣%d\n", area, perimeter);
}
```

In this material, we recommend that C language users migrate to C++, but even when using C++, you can use C's `scanf` and `printf` for input and output.

Ruby

```
a,b = gets.split("␣").map(&:to_i)
area = a*b
perimeter = 2*(a+b)
print sprintf("%d␣%d", area, perimeter)
```

After saving the above content as `rectangle.rb`, etc., execute it on the terminal.

```
$ ruby rectangle.rb
\textit{3 5} # Input from the keyboard
15 16 # Program output
```

In the case of Java, due to the restrictions of the online judge system, it is always necessary to write the answer in a class called `Main`. For that purpose, it is necessary to save it with the file name `Main.java`, so create a folder for each problem and work there.

Java

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int a = scanner.nextInt(), b = scanner.nextInt();
        int area = a*b;
        int perimeter = 2*(a+b);
        System.out.println(area+"␣"+perimeter);
    }
}
```

```
$ javac Main.java
$ java Main
\textit{3 5} # Input from the keyboard
15 16 # Program output display
```

1.7 Practical Notes

1.7.1 Saving Folders

In each chapter, we will handle multiple code snippets for samples and functionality verification. Therefore, to avoid confusion, it is good practice to create a and save files with different names for each theme. And keep each of them working. On the other hand, we do not recommend adding to a file you have already created, making it one huge file. If you do that, it will be difficult to compare the results of running two different codes later.

Here is an example of how to use the ”Terminal” (in the case of MacOSX):

```
$ mkdir programming
# (Create a folder, only the first time)
$ mkdir programming/chapter1
# (Do this for each chapter)
$ cd programming/chapter1
# (Change the current directory.  Save the source code under
$HOME/programming/chapter1/)
```

1.7.2 Testing with Files Using Standard Input/Output and Redirection

This section introduces *how to read data as long as it exists* and *how to test using files*. It is not necessary to master the contents of this section immediately, but it is desirable to learn them in the process of working on Chapters 2 and 3.

When dealing with large inputs and outputs, it is not appropriate to enter them manually from the keyboard. (Manual input carries the risk of typos. Copying and pasting improves this somewhat, but there are limits to the amount of data, and there is still room for errors in selection. When verifying the correctness of a program, it is desirable to eliminate other uncertain factors 100% and focus on the program itself.)

Each problem handles standard input and output, and the correctness of the program is determined by the output for the input. In many cases, the input is read and processed as long as there is input data. So, let’s start with an example. The following environment is basically assumed to be MacOSX, but it should also work on Ubuntu, Cygwin, etc.

Example Problem

Create a program that reads a year, determines if it is a leap year, and outputs the number of days.

(Rough) Example program (C++): leap.cc

C++

```
// Is it every 4 years?
#include <iostream>
using namespace std;
int main() {
  int year;
  while (cin >> year) { // cin is automatically converted to bool, so loop as long as input can be read
   if (year % 4 == 0)
    cout << 366 << endl;
   else
    cout << 365 << endl;
  }
}
```

Compilation (As mentioned above, the $ symbol indicates a command input to the terminal)

In the case of C++:

```
$ g++ -Wall leap.cc
```

In the case of Java:

```
$ javac Main.java
```

Execution Example Italics indicate input from the keyboard. To terminate, press Ctrl and type c or d. This operation is denoted as ˆC or ˆD.

```
$ ./a.out
\textit{2004}
366
\textit{1999}
365
\textit{1900}
```

```
366
\textit{2000}
366
\^{}D
```

Testing with Files Typing on the keyboard every time you run the program is cumbersome, so we want to automate it. Therefore, we will explain testing using redirection and files. It is desirable to learn this at an early stage.

Create correct input and output examples with an editor, and check the contents with the `cat` command:

```
$ cat years.input
2004
1999
1900
2000
$ cat years.output
366
365
365
366
```

Execution using (reading from a file instead of keyboard input):

```
$ ./a.out < years.input
366
365
366
366
```

Saving the execution result to a file (writing to a file instead of displaying it on the screen):

```
$ ./a.out < years.input > test-output
$ cat test-output
366
365
366
366
```

Automatic comparison using :

```
$ diff -u test-output years.output
--- years.output        Fri Oct 14 10:53:52 2005
+++ test-output Fri Oct 14 10:53:56 2005
@@ -1,4 +1,4 @@
 366
 365
-365
+366
 366
```

It tells us that the 4th line is different.

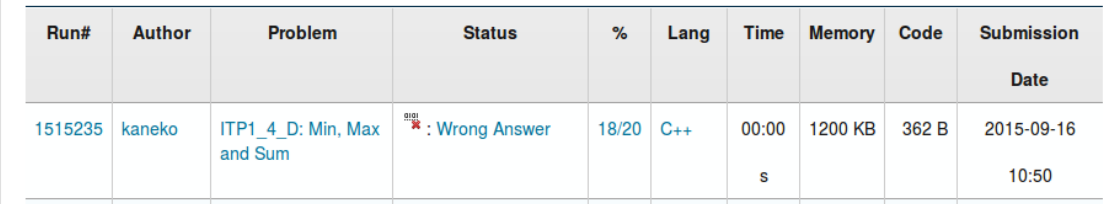
References:

- HWB 15 Commands
  http://hwb.ecc.u-tokyo.ac.jp/wp/information-2/cui/
- HWB 14.4 File Operations Using Commands
  http://hwb.ecc.u-tokyo.ac.jp/wp/information-2/filesystem/cui-fs/

1.7.3 Downloading Judge Data and Running it Locally

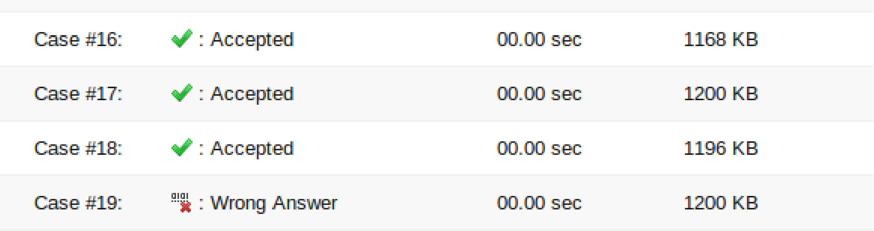
When you submit a program, it may result in , , or instead of Accepted.

Suppose you are solving a problem called ITP1_4_D and you get a Wrong Answer.



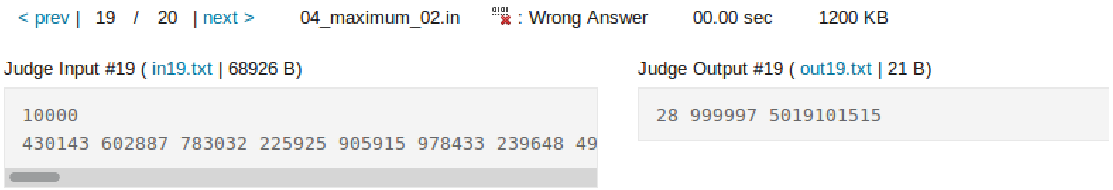
Run#	Author	Problem	Status	%	Lang	Time	Memory	Code	Submission Date
1515235	kaneko	ITP1_4_D: Min, Max and Sum	✗ : Wrong Answer	18/20	C++	00:00 s	1200 KB	362 B	2015-09-16 10:50

The number 18/20 near the center means that you answered correctly up to the 18th test case and failed on the 19th. You can click on that part, which is a hyperlink, to see the details.



Case #16:	✓ : Accepted	00.00 sec	1168 KB
Case #17:	✓ : Accepted	00.00 sec	1200 KB
Case #18:	✓ : Accepted	00.00 sec	1196 KB
Case #19:	✗ : Wrong Answer	00.00 sec	1200 KB

Furthermore, if you click on the line `Case #19:`, you can see the actual data (depending on the problem).



< prev | 19 / 20 | next > 04_maximum_02.in : Wrong Answer 00.00 sec 1200 KB

Judge Input #19 (in19.txt | 68926 B)

```
10000
430143 602887 783032 225925 905915 978433 239648 49
```

Judge Output #19 (out19.txt | 21 B)

```
28 999997 5019101515
```

Let's try this data locally. Since the data is too large to copy and paste, first click on the `in19.txt` part to download it. Depending on your environment and browser, it will be saved in your download folder with a name like `ITP1_4_D_in19.txt`. Execute it by redirection as appropriate.

```
$ ./a.out < ~/Downloads/ITP1_4_D_in19.txt
28 999997 724134219
```

Also, this time it is obvious at a glance that there is a difference between the program's output and the Judge Output, but in general, it is difficult to compare numbers with many digits like 5019101515 by eye (you may not notice even if one character is different), so it is better to download it and compare it with the `diff` command.

Save the execution result to a file (write to a file instead of displaying it on the screen):

```
$ ./a.out < sample-input.txt > my-output.txt
$ cat my-output.txt
1 17 37
```

Compare the two files:

```
$ diff -u sample-output.txt my-output.txt
```

(Output will only be displayed if there are differences)

Part I

Introductory Course

Chapter 2

Input/Output, Arrays, and Simulation

Overview

As a foundation for various problem-solving approaches, we begin with problems that involve examining input data sequentially, and then cover arrays and their basic operations. We also emphasize developing a style of creating and testing programs in parts, rather than all at once. Furthermore, we will experience how the amount of data necessitates the use of appropriate algorithms.

2.1 Maximum and Minimum Values

Example **ICPC Score Totalizer Software** (National Preliminary 2007)

Calculate the average value of a sequence of numbers, excluding the maximum and minimum values.

The input consists of several datasets, each corresponding to a contestant's performance. The number of datasets is 20 or less. The first line of each dataset is the number of judges n ($3 \leq n \leq 100$) who scored the performance. The following n lines each contain a score s ($0 \leq s \leq 1000$) given by each judge. Both n and each s are integers. There are no characters other than digits to represent these numbers in the input. ~~The names of the judges are kept secret.~~ The end of the input is indicated by a single zero on a line.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1147&lang=jp`

Interpreting the “Sample Input” sequentially, we can see that it consists of four datasets:

- (N=) 3 (S=) 1000 342 0
- (N=) 5 (S=) 2 2 9 11 932
- (N=) 5 (S=) 300 1000 0 200 400
- (N=) 8 (S=) 353 242 402 274 283 132 402 523
- 0 (End of input)

```
#--------
3 # There are 3 data points, N=3
1000 # Maximum value
342
0 # Minimum value
#--------
5 # There are 5 data points, N=5
2 # Minimum value
2
9
11
932 # Maximum value
#--------
5 # There are 5 data points
300
1000 # Maximum value
0 # Minimum value
200
400
#--------
8 # There are 8 data points
353
242
402
274
283
132
402
523
#--------
```

```
0 # This is the end
```

Similarly, interpreting the “Sample Output”, we can see that one number is output for each of the above datasets.

- 342 (as is)
- 7 (average of 2, 9, 11)
- 300 (average of 300, 200, 400)
- 326 (average of ...)

2.1.1 Creating the Program

Below are programs that calculate the sum of the judges’ scores and the maximum score. (Since the problem does not ask for the sum or maximum value, these are not direct answers to the problem.) If necessary, use these programs as a reference to create an answer to the problem.

C++

```
\begin{verbatim}
#include <iostream>
using namespace std;
int N, S;
int main() {
    while (cin >> N && N>0) {
        int sum = 0;
        for (int i=0; i<N; ++i) {
            cin >> S;
            sum += S; // std::accumulate can also be used
        }
        cout << sum << endl;
    }
}
\end{verbatim}
```

C++

```
\begin{verbatim}
// (Partially omitted)
    while (cin >> N && N>0) {
        int largest = 0;
        for (int i=0; i<N; ++i) { // std::max_element can also be used
            cin >> S;
            if (largest < S) largest = S;
        }
        cout << largest << endl;
    }
\end{verbatim}
```

When transitioning from C to C++, it is good to learn and be able to use the shaded parts (since the basics are the same). See Section 1.7 for how to compile.

C Language Prohibited

When reading this material, C language is insufficient, and it is necessary to use some of the features of C++. We will introduce the necessary parts little by little, so get used to `iostream`, `cin`, `cout` etc. at this point.

Use Functions

You can use functions for `max`, `min`, `sum`, etc.

Python3

```
\begin{verbatim}
while True:
    N = int(input())
    if N == 0:
        break
    S = []
    for i in range(N):
        S.append(int(input()))
    print(sum(S))
    print(max(S))
\end{verbatim}
```

Note that to perform integer division in Python3, use the `//` operator instead of `/`.

2.1.2 Testing the Program

Testing with Sample Input

Let’s verify that the output for the “Sample Input” in the problem statement matches.

Testing with Judge Data

For this problem, the secret input and output used by the judge are publicly available. By using these, you can further test your program for errors on your own. (You might discover that your program behaves correctly for the “Sample Input” but incorrectly for other data.)

- Input `http://www.logos.ic.i.u-tokyo.ac.jp/icpc2007/jp/domestic/datasets/A/A1`
- Output `http://www.logos.ic.i.u-tokyo.ac.jp/icpc2007/jp/domestic/datasets/A/A1.ans`

To check if the output matches, use the `diff` command. See Section 1.7 for how to execute it.

```
\$ python3 icpc.py < A1 > my-out.txt
\$ diff -u my-out.txt A1.ans
```

```
\$ ./a.out < A1 > my-out.txt
\$ diff -u my-out.txt A1.ans
```

Do Not Postpone

At this point (before moving on to the next chapter), be sure to learn how to use redirection and the `diff` command.

2.2 Estimating Computation Time and Number of Trials

A solution method that computers excel at is trying all possibilities by brute force. In fact, many problems can be solved this way. A solution method that computers excel at is trying all possibilities by brute force. In fact, many problems can be solved this way. To enumerate all possibilities, we use for loops or recursion.

Problem **Tax Rate Changed** (National Preliminary 2014)

Given the total price of two items including tax before a tax rate change, create a program to calculate the maximum possible total price including tax with the new tax rate. Refer to the problem statement for detailed conditions such as rounding down to the nearest yen.

(Problem continued)

Constraints (excerpt): $10 < s < 1000$, the prices of the items before tax range from 1 yen to $s - 1$ yen.

Time Limit: 8 sec, Memory Limit: 65536 KB

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1192&lang=jp

Let's try creating a program with the following approach: (1) Try all combinations of possible prices for the two items. (2) Check if the combination results in the given total price with the old tax rate. If not, discard it. (3) Calculate the total price with the new tax rate. If it's the maximum value, store it.

Assuming there is a function `tax(rate, price_without_tax)` to calculate the price including tax, the structure would be as follows. In the problem statement, a lowercase s is used, but in the programs in this document, we use uppercase to distinguish it from variables like i, j, k, and to mark it as originating from the problem statement.

C++

```cpp
int maximum = 0;
for (int i=1; i<S; ++i)
  for (int j=i; j<S; ++j)
    if (tax(X,i)+tax(X,j) == S) // (*)
      maximum = max(maximum, tax(Y,i)+tax(Y,j));
```

It is also possible to consider ways to reduce the number of iterations for this example problem, but it is not essential for C++ this time. In the case of Python, some ingenuity is required. In the code below, we use the property that if the total price including tax of a pair (a_0, b_0) exceeds S, then the total price including tax of a pair (a_0, b) with $b > b_0$ will also necessarily exceed S, so it is not necessary to consider it.

Function Required (No Cheating)

Be sure to create the function `tax`. The ability to create functions is important when dealing with increasingly complex programs. Acquire this skill at this point.

Python3

```python
\begin{verbatim}
def tax(p,x):
    return p*(100+x) // 100      # Integer division
def solve(X,Y,S):
    for a in range(1,S):
        for b in range(1,S):
```

```
 7                sum = tax(a,X)+tax(b,X)
 8                if sum == S:
 9                    # Consider tax(a,Y)+tax(b,Y) with the new tax
10                if sum > S:
11                    break           # If b increases, sum increases
12        return best
13    while True:
14        X,Y,S = map(int, input().strip().split('␣'))
15        if X == 0:
16            break
17        print(solve(X,Y,S))
18    \end{verbatim}
```

TLE (Time limit exceeded)

Since Python is slower than C++, you may not get AC with the same approach as C++. For example, in the sample code above, lines 9 and 10 have been optimized for speed.

Tax Included vs. Tax Excluded

In this material, we created a function to calculate the price including tax from the price excluding tax and the tax rate. What about the opposite approach, calculating the price excluding tax from the price including tax? In fact, that approach has many pitfalls and is not recommended (if you are interested, think about the reason after getting AC with the recommended method).

Input of Multiple Datasets In this problem, multiple datasets may be given. That is, while data is being given, it is necessary to read it, output the solution, and when the data ends, terminate the program. Using the condition ”The end of the input is indicated by a line consisting of three zeros separated by spaces” and the fact that X is positive in normal datasets, we recommend writing the program with the following structure.

C++

```
\begin{verbatim}
#include <iostream>
using namespace std;
int X, Y, S;
int solve() {
  ... // Calculate the maximum value for X, Y, S
}
int main() {
    while (cin >> X >> Y >> S && X>0) {
```

```
10          cout << solve() << endl;
11      }
12  }
13  \end{verbatim}
```

The `cin >> X >> Y >> S` part of the `while` statement's condition returns a reference to `cin`, and when cast to `bool`, it returns whether `cin` is in a normal state, i.e., whether `X, Y, S` were read successfully. If the input file is incorrect (a common case is a typo or giving input from a different problem), this will become `false` and the program will terminate. If it was read successfully, `X, Y, S` could be either the `X, Y, S` of a dataset to be processed, or three zeros separated by spaces, which is the end-of-input marker. In the former case, it should be a positive integer, so we test if it is 0 to distinguish.

Submission to AOJ Create a program based on the above approach and confirm that it is accepted by AOJ.

If you do not get accepted, you can verify it manually. First, download the input (`http://icpc.iisf.or.jp/past-icpc/domestic2014/qualify14_ans/A1`) and the correct answer (`http://icpc.iisf.or.jp/past-icpc/domestic2014/qualify14_ans/A1.ans`).

```
\$ ./a.out < A1 > my-output.txt
\$ diff -u A1.ans my-output.txt
```

If there is a difference, the line numbers will be displayed.

Simple Consideration of Computational Complexity (estimation of the number of trials) Many contest problems have time limits, so it is sometimes necessary to estimate the execution time and devise a solution that will finish in time. Also, even in programs used for practical purposes outside of contests, there are often requirements regarding execution speed. If you find out that there is a problem with the speed after writing the program, it may be difficult to rewrite, so it is desirable to have some idea *before* writing the program.

Since real computers are complex devices, it is not easy to accurately predict the execution time. Therefore, we consider a guideline based on a simple model. For example, we count how many times basic operations such as addition, subtraction, multiplication, and division are performed in the process of executing the program. This model ignores details such as the fact that addition and division may have different speeds, or that two instructions may be executed simultaneously. It is a guideline, so the correspondence with reality needs to be discussed separately, but it is often useful.

Let's consider how many times line 4 (*) in the above program is executed at most (it is not necessary to find it exactly). Let's denote the number of times (since it depends on S) as $f(S)$. Since the `for` loop on line 2 repeats S times, and the `for` loop on line 3 repeats at most S times, $f(S) \leq S^2 \leq 1\,000^2 = 10^6$.

Applying this number (10^6) to Table 3.1, we can see that it is possible to execute it in 1 second with a margin (assuming that the tax function is implemented efficiently). [1] The numerical values in the table need to be measured

[1] Note: This estimate is for one dataset. On the other hand, the entire problem requires answering ”input consists of multiple datasets” within 8

Table 2.1: Guideline for the number of operations executable in 1 second in C++ (adapted and reprinted from [3])

1 000 000	Possible with margin
10 000 000	Probably possible
100 000 000	Possible with very simple operations

empirically according to the execution environment. As a guideline, since the CPUs of many recent computers operate at about 1 GHz ($= 10^9$), if the CPU can execute an operation within 100 cycles, it can be expected to execute 10^6 times per second.

In the scope of this material, it is mostly okay to trust this table, but for accurate prediction, perform calibration. That is, after estimating how many basic operations the created solution performs for a numerical value N corresponding to the input, experiment with inputs corresponding to several N and correlate them with the actual computation time in seconds. For example, memory access and new/delete may be 10 times and 100 times slower than arithmetic operations, respectively. Also, in languages other than C and C++, execution often takes longer. Consider the time limit for each problem and the hardware of the online judge.

Python is slower than C++

Depending on the problem, it is good to estimate with a margin of 20-200 times. Since the time limit in online judges is not necessarily appropriate for Python, it is okay if you can download the judge data and confirm the correct answer.

Problem **Space Coconut Grab** (Mock National Preliminary 2007)

Find the location where space coconut crabs appear. Given that energy E has been observed, the candidate locations are integer coordinates (x, y, z) that satisfy $x + y^2 + z^3 = E$, and the location is limited to the one with the minimum value of x+y+z. Find the minimum value of x+y+z. (x, y, z are non-negative integers, E is a positive integer less than or equal to 1,000,000, and space coconut crabs are the largest crustaceans in space, with a body length of 400 meters or more after growth, and a leg span of 1,000 meters or more)

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2012&lang=jp`

First, consider whether trying all possibilities will finish in time. (If it does, that is the easiest program to implement)

seconds, so the computation time needs to be estimated considering the upper limit of the number of datasets. If there is no description about the number of datasets, it is strictly a defect in the problem, but it is okay to assume about 10.

Considering the range of variables x, y, and z, they are x:[0,E], y:[0,$\sqrt{E}$], z:[0,$^{3}\sqrt{E}$], respectively. The number of combinations of them (x, y, z) is at most $10^6 \cdot 10^3 \cdot 10^2 = 10^{11}$, since the maximum value of E is 1,000,000= 10^6. Even considering the time limit of 8 seconds, this approach does not seem to finish in time.

As a guideline for reducing the number of trials, it is not necessary to consider all combinations of (x, y, z), but only the range that satisfies $x + y^2 + z^3 = E$. For example, if x and y are determined, z can be calculated from E (it is okay to ignore cases where z is not an integer). The number of combinations to check with this approach is the number of combinations of (x, y), which is $10^6 \cdot 10^3$. This value is still large, but it has been reduced to 1/100 compared to the previous one. Similarly, you can determine x and z and find y, or determine z and y and find x. Find the number of combinations to check in those approaches. Confirm that the smallest one is within the time limit, and actually implement it and submit it to AOJ.

Python–TLE

Unfortunately, the above approach exceeds the time limit (TLE) in Python. As a countermeasure, it is sufficient to avoid the double loop and use only a single loop for `z`. The loop for `y` can be omitted due to monotonicity.

Python3

```
\begin{verbatim}
import math
# ...(omitted)...
y = int(math.floor(math.sqrt(E-z**3)+1e-7))
\end{verbatim}
```

Chapter 3

Input/Output, Arrays, and Simulation

Overview

As a foundation for various problem-solving approaches, we begin with problems that involve examining input data sequentially, and then cover arrays and their basic operations. We also emphasize developing a style of creating and testing programs in parts, rather than all at once. Furthermore, we will experience how the amount of data necessitates the use of appropriate algorithms.

3.1 Maximum and Minimum Values

Example **ICPC Score Totalizer Software** (National Preliminary 2007)

Calculate the average value of a sequence of numbers, excluding the maximum and minimum values.

The input consists of several datasets, each corresponding to a contestant's performance. The number of datasets is 20 or less. The first line of each dataset is the number of judges n ($3 \leq n \leq 100$) who scored the performance. The following n lines each contain a score s ($0 \leq s \leq 1000$) given by each judge. Both n and each s are integers. There are no characters other than digits to represent these numbers in the input. The end of the input is indicated by a single zero on a line.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1147&lang=jp`

Interpreting the "Sample Input" sequentially, we can see that it consists of four datasets:

- (N=) 3 (S=) 1000 342 0
- (N=) 5 (S=) 2 2 9 11 932
- (N=) 5 (S=) 300 1000 0 200 400
- (N=) 8 (S=) 353 242 402 274 283 132 402 523
- 0 (End of input)

```
#--------
3 # There are 3 data points, N=3
1000 # Maximum value
342
0 # Minimum value
#--------
5 # There are 5 data points, N=5
2 # Minimum value
2
9
11
932 # Maximum value
#--------
5 # There are 5 data points
300
1000 # Maximum value
0 # Minimum value
200
400
#--------
8 # There are 8 data points
353
242
402
274
283
132
402
523
#--------
0 # This is the end
```

Similarly, interpreting the “Sample Output”, we can see that one number is output for each of the above datasets.

- 342 (as is)
- 7 (average of 2, 9, 11)
- 300 (average of 300, 200, 400)
- 326 (average of ...)

3.1.1 Creating the Program

Below are programs that calculate the sum of the judges’ scores and the maximum score. (Since the problem does not ask for the sum or maximum value, these are not direct answers to the problem.) If necessary, use these programs as a reference to create an answer to the problem.

C++

```
\begin{verbatim}
#include <iostream>
using namespace std;
int N, S;
int main() {
    while (cin >> N && N>0) {
        int sum = 0;
        for (int i=0; i<N; ++i) {
            cin >> S;
            sum += S; // std::accumulate can also be used
        }
        cout << sum << endl;
    }
}
\end{verbatim}
```

C++

```
\begin{verbatim}
// (Partially omitted)
    while (cin >> N && N>0) {
        int largest = 0;
        for (int i=0; i<N; ++i) { // std::max_element can also be used
            cin >> S;
            if (largest < S) largest = S;
        }
        cout << largest << endl;
    }
\end{verbatim}
```

When transitioning from C to C++, it is good to learn and be able to use the shaded parts (since the basics are the same). See Section 1.7 for how to compile.

C Language Prohibited

When reading this material, C language is insufficient, and it is necessary to use some of the features of C++. We will introduce the necessary parts little by little, so get used to `iostream,cin,cout` etc. at this point.

Use Functions

You can use functions for `max,min,sum`, etc.

Python3

```
\begin{verbatim}
while True:
    N = int(input())
    if N == 0:
        break
    S = []
    for i in range(N):
        S.append(int(input()))
    print(sum(S))
    print(max(S))
\end{verbatim}
```

Note that to perform integer division in Python3, use the `//` operator instead of `/`.

3.1.2 Testing the Program

Testing with Sample Input

Let’s verify that the output for the “Sample Input” in the problem statement matches.

Testing with Judge Data

For this problem, the secret input and output used by the judge are publicly available. By using these, you can further test your program for errors on your own. (You might discover that your program behaves correctly for the "Sample Input" but incorrectly for other data.)

- Input `http://www.logos.ic.i.u-tokyo.ac.jp/icpc2007/jp/domestic/datasets/A/A1`
- Output `http://www.logos.ic.i.u-tokyo.ac.jp/icpc2007/jp/domestic/datasets/A/A1.ans`

To check if the output matches, use the `diff` command. See Section 1.7 for how to execute it.

```
\$ python3 icpc.py < A1 > my-out.txt
\$ diff -u my-out.txt A1.ans
```

```
\$ ./a.out < A1 > my-out.txt
\$ diff -u my-out.txt A1.ans
```

STOP Do Not Postpone

At this point (before moving on to the next chapter), be sure to learn how to use redirection and the `diff` command.

3.2 Estimating Computation Time and Number of Trials

A solution method that computers excel at is trying all possibilities by brute force. In fact, many problems can be solved this way. A solution method that computers excel at is trying all possibilities by brute force. In fact, many problems can be solved this way. To enumerate all possibilities, we use for loops or recursion.

Problem **Tax Rate Changed** (National Preliminary 2014)

Given the total price of two items including tax before a tax rate change, create a program to calculate the maximum possible total price including tax with the new tax rate. Refer to the problem statement for detailed conditions such as rounding down to the nearest yen.

(Problem continued)

Constraints (excerpt): $10 < s < 1000$, the prices of the items before tax range from 1 yen to $s - 1$ yen.

Time Limit: 8 sec, Memory Limit: 65536 KB

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1192&lang=jp

Let's try creating a program with the following approach: (1) Try all combinations of possible prices for the two items. (2) Check if the combination results in the given total price with the old tax rate. If not, discard it. (3) Calculate the total price with the new tax rate. If it's the maximum value, store it.

Assuming there is a function `tax(rate, price_without_tax)` to calculate the price including tax, the structure would be as follows. In the problem statement, a lowercase s is used, but in the programs in this document, we use uppercase to distinguish it from variables like i, j, k, and to mark it as originating from the problem statement.

C++

```cpp
  int maximum = 0;
  for (int i=1; i<S; ++i)
    for (int j=i; j<S; ++j)
      if (tax(X,i)+tax(X,j) == S) // (*)
        maximum = max(maximum, tax(Y,i)+tax(Y,j));
```

It is also possible to consider ways to reduce the number of iterations for this example problem, but it is not essential for C++ this time. In the case of Python, some ingenuity is required. In the code below, we use the property that if the total price including tax of a pair (a_0, b_0) exceeds S, then the total price including tax of a pair (a_0, b) with $b > b_0$ will also necessarily exceed S, so it is not necessary to consider it.

Function Required (No Cheating)

Be sure to create the function `tax`. The ability to create functions is important when dealing with increasingly complex programs. Acquire this skill at this point.

Python3

```python
\begin{verbatim}
def tax(p,x):
    return p*(100+x) // 100      # Integer division
def solve(X,Y,S):
    for a in range(1,S):
        for b in range(1,S):
```

```
7               sum = tax(a,X)+tax(b,X)
8               if sum == S:
9                   # Consider tax(a,Y)+tax(b,Y) with the new tax
10              if sum > S:
11                  break           # If b increases, sum increases
12      return best
13  while True:
14      X,Y,S = map(int, input().strip().split('␣'))
15      if X == 0:
16          break
17      print(solve(X,Y,S))
18  \end{verbatim}
```

TLE (Time limit exceeded)

Since Python is slower than C++, you may not get AC with the same approach as C++. For example, in the sample code above, lines 9 and 10 have been optimized for speed.

Tax Included vs. Tax Excluded

In this material, we created a function to calculate the price including tax from the price excluding tax and the tax rate. What about the opposite approach, calculating the price excluding tax from the price including tax? In fact, that approach has many pitfalls and is not recommended (if you are interested, think about the reason after getting AC with the recommended method).

Input of Multiple Datasets In this problem, multiple datasets may be given. That is, while data is being given, it is necessary to read it, output the solution, and when the data ends, terminate the program. Using the condition ”The end of the input is indicated by a line consisting of three zeros separated by spaces” and the fact that X is positive in normal datasets, we recommend writing the program with the following structure.

C++

```
\begin{verbatim}
#include <iostream>
using namespace std;
int X, Y, S;
int solve() {
  ... // Calculate the maximum value for X, Y, S
}
int main() {
    while (cin >> X >> Y >> S && X>0) {
```

```
10          cout << solve() << endl;
11      }
12  }
13  \end{verbatim}
```

The `cin >> X >> Y >> S` part of the `while` statement's condition returns a reference to `cin`, and when cast to `bool`, it returns whether `cin` is in a normal state, i.e., whether `X, Y, S` were read successfully. If the input file is incorrect (a common case is a typo or giving input from a different problem), this will become `false` and the program will terminate. If it was read successfully, `X, Y, S` could be either the `X, Y, S` of a dataset to be processed, or three zeros separated by spaces, which is the end-of-input marker. In the former case, it should be a positive integer, so we test if it is 0 to distinguish.

Submission to AOJ Create a program based on the above approach and confirm that it is accepted by AOJ.

If you do not get accepted, you can verify it manually. First, download the input (http://icpc.iisf.or.jp/past-icpc/domestic2014/qualify14_ans/A1) and the correct answer (http://icpc.iisf.or.jp/past-icpc/domestic2014/qualify14_ans/A1.ans).

```
\$ ./a.out < A1 > my-output.txt
\$ diff -u A1.ans my-output.txt
```

If there is a difference, the line numbers will be displayed.

Simple Consideration of Computational Complexity (estimation of the number of trials) Many contest problems have time limits, so it is sometimes necessary to estimate the execution time and devise a solution that will finish in time. Also, even in programs used for practical purposes outside of contests, there are often requirements regarding execution speed. If you find out that there is a problem with the speed after writing the program, it may be difficult to rewrite, so it is desirable to have some idea *before* writing the program.

Since real computers are complex devices, it is not easy to accurately predict the execution time. Therefore, we consider a guideline based on a simple model. For example, we count how many times basic operations such as addition, subtraction, multiplication, and division are performed in the process of executing the program. This model ignores details such as the fact that addition and division may have different speeds, or that two instructions may be executed simultaneously. It is a guideline, so the correspondence with reality needs to be discussed separately, but it is often useful.

Let's consider how many times line 4 (*) in the above program is executed at most (it is not necessary to find it exactly). Let's denote the number of times (since it depends on S) as $f(S)$. Since the `for` loop on line 2 repeats S times, and the `for` loop on line 3 repeats at most S times, $f(S) \leq S^2 \leq 1\,000^2 = 10^6$.

Applying this number (10^6) to Table 3.1, we can see that it is possible to execute it in 1 second with a margin (assuming that the tax function is implemented efficiently). [1] The numerical values in the table need to be measured

[1] Note: This estimate is for one dataset. On the other hand, the entire problem requires answering ”input consists of multiple datasets” within 8

Table 3.1: Guideline for the number of operations executable in 1 second in `C++` (adapted and reprinted from [3])

1 000 000	Possible with margin
10 000 000	Probably possible
100 000 000	Possible with very simple operations

empirically according to the execution environment. As a guideline, since the CPUs of many recent computers operate at about 1 GHz ($= 10^9$), if the CPU can execute an operation within 100 cycles, it can be expected to execute 10^6 times per second.

In the scope of this material, it is mostly okay to trust this table, but for accurate prediction, perform calibration. That is, after estimating how many basic operations the created solution performs for a numerical value N corresponding to the input, experiment with inputs corresponding to several N and correlate them with the actual computation time in seconds. For example, memory access and new/delete may be 10 times and 100 times slower than arithmetic operations, respectively. Also, in languages other than C and C++, execution often takes longer. Consider the time limit for each problem and the hardware of the online judge.

Python is slower than C++

Depending on the problem, it is good to estimate with a margin of 20-200 times. Since the time limit in online judges is not necessarily appropriate for Python, it is okay if you can download the judge data and confirm the correct answer.

Problem **Space Coconut Grab** (Mock National Preliminary 2007)

Find the location where space coconut crabs appear. Given that energy E has been observed, the candidate locations are integer coordinates (x, y, z) that satisfy $x + y^2 + z^3 = E$, and the location is limited to the one with the minimum value of x+y+z. Find the minimum value of x+y+z. (x, y, z are non-negative integers, E is a positive integer less than or equal to 1,000,000, and space coconut crabs are the largest crustaceans in space, with a body length of 400 meters or more after growth, and a leg span of 1,000 meters or more)

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2012&lang=jp`

First, consider whether trying all possibilities will finish in time. (If it does, that is the easiest program to implement)

seconds, so the computation time needs to be estimated considering the upper limit of the number of datasets. If there is no description about the number of datasets, it is strictly a defect in the problem, but it is okay to assume about 10.

Considering the range of variables x, y, and z, they are x:[0,E], y:[0,$\sqrt{E}$], z:[0,$\sqrt[3]{E}$], respectively. The number of combinations of them (x, y, z) is at most $10^6 \cdot 10^3 \cdot 10^2 = 10^{11}$, since the maximum value of E is 1,000,000= 10^6. Even considering the time limit of 8 seconds, this approach does not seem to finish in time.

As a guideline for reducing the number of trials, it is not necessary to consider all combinations of (x, y, z), but only the range that satisfies $x + y^2 + z^3 = E$. For example, if x and y are determined, z can be calculated from E (it is okay to ignore cases where z is not an integer). The number of combinations to check with this approach is the number of combinations of (x, y), which is $10^6 \cdot 10^3$. This value is still large, but it has been reduced to 1/100 compared to the previous one. Similarly, you can determine x and z and find y, or determine z and y and find x. Find the number of combinations to check in those approaches. Confirm that the smallest one is within the time limit, and actually implement it and submit it to AOJ.

Python–TLE

Unfortunately, the above approach exceeds the time limit (TLE) in Python. As a countermeasure, it is sufficient to avoid the double loop and use only a single loop for `z`. The loop for `y` can be omitted due to monotonicity.

Python3

```
\begin{verbatim}
import math
# ...(omitted)...
y = int(math.floor(math.sqrt(E-z**3)+1e-7))
\end{verbatim}
```

3.3 Exercises

Problem **Square Route** (Mock National Preliminary 2007)

In a city with approximately 1500 vertical and horizontal roads, please count the number of squares.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2015&lang=jp`

Note: Trying all combinations of 4 points and checking if they form a square will not finish in time, so a more clever method is needed. Since counting squares one by one would result in too many counts, a method is needed to count squares with certain properties together.

Hint: There are methods that utilize standard data structures, which will be covered in later chapters, and methods that utilize elementary properties of squares (consider squares that share a common line when extending the diagonal).

Chapter 4

Sorting and Greedy Algorithms

Overview

Rearranging data according to a certain criterion is called sorting. Most standard libraries in programming languages provide sorting methods, so first, let's learn how to use them. In this material, we take the position that it is sufficient to obtain the sorted result, but if you are interested in the sorting methods themselves, please refer to, for example, Reference$^{\text{Strategy}}$[2, pp. 51–(Chapter 3)].
Sorting data can also be useful as a tool for problem-solving. These include optimization problems and placement problems, which may not seem related to sorting at first glance.

4.1 Various Sorting Algorithms

4.1.1 Sorting Numbers

In C++ and Ruby, `sort` and `sort!` are provided as standard functions.

C++

```
\begin{verbatim}
#include <algorithm>
#include <iostream>
using namespace std;
int A[5] = {3,5,1,2,4};
int main() {
   sort(A,A+5); // Specifies a half-open interval [l,r).  Equivalent to sort(&A[0], &A[5])
   ... // Try outputting A to cout
}
```

```
\end{verbatim}
```

To specify the range for sorting an array, use a pointer to an element of the array, such as `&A[0]`. In the case of a one-dimensional array, simply writing `A` will automatically convert it to a pointer to the first element. To specify the range of a `vector` or `array` instead of an array, use the notation `A.begin(),A.end()`. These `begin` and `end` are functions that return iterators, which are a generalization of pointers, and point to the locations as their names suggest. You can also specify a partial range, such as `A.begin()+2,A.begin()+5`. Furthermore, in C++11 and later, you can use `begin(A)` to handle both arrays and vectors in a common way.

Python3

```python
\begin{verbatim}
a = [3,5,1,2,4]
a.sort()
a
# [1, 2, 3, 4, 5]
\end{verbatim}
```

Example **Sort II** (AOJ)

Create a program that sorts a given n numbers in ascending order and outputs them.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=10029&lang=jp

Example solution:

C++

```cpp
\begin{verbatim}
int N, A[1000000+10];
int main() {
    cin >> N;
    for (int i=0; i<N; ++i) cin >> A[i]; // Input A
    ... // Sort A
    for (int i=0; i<N; ++i) cout << (i?"␣":"") << A[i]; // Output A
    cout << endl;
}
\end{verbatim}
```

The `(i?" ":"")` in the output part is a fine adjustment to insert a space only between elements (no space for the first element `i==0`).

4.1.2 Sorting Numbers

In C++ and Ruby, `sort` and `sort!` are provided as standard functions.

C++

```
\begin{verbatim}
#include <algorithm>
#include <iostream>
using namespace std;
int A[5] = {3,5,1,2,4};
int main() {
   sort(A,A+5); // Specifies a half-open interval [l,r).  Equivalent to sort(&A[0], &A[5])
   ... // Try outputting A to cout
}
\end{verbatim}
```

To specify the range for sorting an array, use a pointer to an element of the array, such as `&A[0]`. In the case of a one-dimensional array, simply writing `A` will automatically convert it to a pointer to the first element. To specify the range of a `vector` or `array` instead of an array, use the notation `A.begin(),A.end()`. These `begin` and `end` are functions that return iterators, which are a generalization of pointers, and point to the locations as their names suggest. You can also specify a partial range, such as `A.begin()+2,A.begin()+5`. Furthermore, in C++11 and later, you can use `begin(A)` to handle both arrays and vectors in a common way.

Python3

```
\begin{verbatim}
a = [3,5,1,2,4]
a.sort()
a
# [1, 2, 3, 4, 5]
\end{verbatim}
```

Example **Sort II** (AOJ)

Create a program that sorts a given n numbers in ascending order and outputs them.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=10029&lang=jp

Example solution:

C++

```
\begin{verbatim}
int N, A[1000000+10];
int main() {
    cin >> N;
    for (int i=0; i<N; ++i) cin >> A[i]; // Input A
    ... // Sort A
    for (int i=0; i<N; ++i) cout << (i?"␣":"") << A[i]; // Output A
    cout << endl;
}
\end{verbatim}
```

The `(i?" ":"")` in the output part is a fine adjustment to insert a space only between elements (no space for the first element `i==0`).

4.1.3 Sorting Strings

Example **Finding Minimum String** (AOJ)

Find the lexicographically smallest string among N strings consisting only of lowercase alphabets. Although not specified in the problem statement, N does not exceed 1000.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=10021&lang=jp

Note: Lexicographical order differs from the comparison order of `std::string` in C++ when uppercase and lowercase letters are mixed. (However, since this time it is only lowercase, there is no need to worry about that).

C++

```
\begin{verbatim}
#include <algorithm> // For sort
#include <string> // For string (character string)
#include <iostream>
using namespace std;
string A[1000];
int N;
int main() {
    cin >> N;
    if (N > 1000) abort();
    for (int i=0; i<N; ++i) cin >> A[i];
    ... // Sort A in the same way as for integers
    cout << A[0] << endl;
}
\end{verbatim}
```

4.1.4 Pairs and Sorting

Pairs

We introduce the representation of pairs (or tuples). Many real-world pieces of information are naturally represented as pairs, such as ⟨start time, end time⟩, ⟨height, weight⟩, or ⟨student ID, score⟩.

C++

```
\begin{verbatim}
#include <utility> // For pair
#include <iostream>
using namespace std;
int main() {
  pair<int,int> a(2,4); // A pair of integers
  cout << a.first << '␣' << a.second << endl; // Displays 2 4

  a.first = 3;
  a.second = 5;
  cout << a.first << '␣' << a.second << endl; // Displays 3 5

  a = make_pair(10, -30);
  cout << a.first << '␣' << a.second << endl; // Displays 10 -30

  pair<double,char> b; // A pair of a double and a character
  b.first = 0.5;
  b.second = 'X';
  cout << b.first << '␣' << b.second << endl; // Displays 0.5 X
}
\end{verbatim}
```

In Python and Ruby, lists (arrays or lists) can be easily used, so lists are used (without particularly distinguishing pairs).

Arrays of Pairs and Sorting

Next, we handle arrays of pairs. For example, when representing the height and weight of a person as a pair, the data of multiple people's heights and weights can be associated with an array of pairs. Just as we sorted an array of integers (`sort`) previously, we can also sort an array of pairs. By default, if the first elements are different, the order is determined by the first element, and when the first elements are the same, the second elements are compared.

C++

```
\begin{verbatim}
#include <utility> // For pair
#include <algorithm> // For sort
#include <iostream>
using namespace std;
int main() {
  pair<int,int> a[3]; // Array of integer pairs
  a[0] = make_pair(170,60);
  a[1] = make_pair(180,90);
  a[2] = make_pair(170,65);

  for (int i=0; i<3; ++i) // Display from a[0] to a[2]
    cout << a[i].first << '␣' << a[i].second << endl;
  // Should display:
  // 170 60
  // 180 90
  // 170 65

  sort(a, a+3); // Sort from a[0] to a[2]

  for (int i=0; i<3; ++i) // Display from a[0] to a[2]
    cout << a[i].first << '␣' << a[i].second << endl;
  // Should display:
  // 170 60
  // 170 65
  // 180 90
}
\end{verbatim}
```

Python3

```
\begin{verbatim}
a = [[3,5],[2,9],[3,6]]
print(a) # [[3, 5], [2, 9], [3, 6]]
a.sort()
print(a) # [[2, 9], [3, 5], [3, 6]]
\end{verbatim}
```

Tuples of Three or More Elements

In C++11, the `tuple` type is available to represent tuples of three or more elements.[1] In earlier versions of C++, `pair` was nested, such as `pair<int,pair<int,int> >`, but now it can be written more concisely.

[1] http://en.cppreference.com/w/cpp/utility/tuple

C++11

```
\begin{verbatim}
#include <tuple>
#include <iostream>
#include <algorithm>
using namespace std;
int main() {
  tuple<int,int,int> a = {3,1,4};
  cout << get<0>(a) << "\n";    // 3
  cout << get<1>(a) << "\n";    // 1
  cout << get<2>(a) << "\n";    // 4

  tuple<int,int,int> array[] = {{3,1,4}, {1,5,9}, {2,6,5}};
  sort(array, array+3);

  tuple<int,int,int> t = array[0];
  cout << get<0>(t) << "\n";    // 1
  cout << get<1>(t) << "\n";    // 5
  cout << get<2>(t) << "\n";    // 9
}
\end{verbatim}
```

4.1.5 Sorting in Descending Order

The standard `sort` function sorts in ascending order. How can we sort in descending order?

1. Sort in ascending order first, then reverse the order (this is sufficient for now).

C++

```
\begin{verbatim}
int A[5] = {3,5,1,2,4};
int main() {
   sort(A,A+5);
   // In C++11, sort(begin(A),end(A));
   reverse(A,A+5); // Reverses the order of the given range
   ...  // Try outputting A to cout
}
\end{verbatim}
```

Python3

```
\begin{verbatim}
a = [3,5,1,2,4]
a.sort()
a.reverse() # Reverses the order of a
\end{verbatim}
```

2. Use `reverse_iterator`

C++14

```
\begin{verbatim}
  sort(rbegin(A),rend(A));
\end{verbatim}
```

Python3

```
\begin{verbatim}
a.sort(reverse=True)
\end{verbatim}
```

3. Pass a comparison function (general purpose)

C++11

```
\begin{verbatim}
int A[5] = {3,5,1,2,4};
int main() {
   sort(begin(A),end(A),[](int p, int q){ return p > q; });
   ...  // Try outputting A[i] to cout
}
\end{verbatim}
```

The explanation of the syntax is omitted, but the part `[](int p, int q){ return p > q; }` is an anonymous function that determines the order of two integers.

Python3

```
\begin{verbatim}
a.sort(key=lambda e: -e)
\end{verbatim}
```

4.1.6 Sorting Strings

Example **Finding Minimum String** (AOJ)

Find the lexicographically smallest string among N strings consisting only of lowercase alphabets. Although not specified in the problem statement, N does not exceed 1000.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=10021&lang=jp`

Note: Lexicographical order differs from the comparison order of `std::string` in C++ when uppercase and lowercase letters are mixed. (However, since this time it is only lowercase, there is no need to worry about that).

C++

```
\begin{verbatim}
#include <algorithm> // For sort
#include <string> // For string (character string)
#include <iostream>
using namespace std;
string A[1000];
int N;
int main() {
    cin >> N;
    if (N > 1000) abort();
    for (int i=0; i<N; ++i) cin >> A[i];
    ... // Sort A in the same way as for integers
    cout << A[0] << endl;
}
\end{verbatim}
```

4.1.7 Pairs and Sorting

Pairs

We introduce the representation of pairs (or tuples). Many real-world pieces of information are naturally represented as pairs, such as ⟨start time, end time⟩, ⟨height, weight⟩, or ⟨student ID, score⟩.

C++

```
\begin{verbatim}
#include <utility> // For pair
#include <iostream>
```

```
using namespace std;
int main() {
  pair<int,int> a(2,4); // A pair of integers
  cout << a.first << '␣' << a.second << endl; // Displays 2 4

  a.first = 3;
  a.second = 5;
  cout << a.first << '␣' << a.second << endl; // Displays 3 5

  a = make_pair(10, -30);
  cout << a.first << '␣' << a.second << endl; // Displays 10 -30

  pair<double,char> b; // A pair of a double and a character
  b.first = 0.5;
  b.second = 'X';
  cout << b.first << '␣' << b.second << endl; // Displays 0.5 X
}
\end{verbatim}
```

In Python and Ruby, lists (arrays or lists) can be easily used, so lists are used (without particularly distinguishing pairs).

Arrays of Pairs and Sorting

Next, we handle arrays of pairs. For example, when representing the height and weight of a person as a pair, the data of multiple people's heights and weights can be associated with an array of pairs. Just as we sorted an array of integers (`sort`) previously, we can also sort an array of pairs. By default, if the first elements are different, the order is determined by the first element, and when the first elements are the same, the second elements are compared.

C++

```
\begin{verbatim}
#include <utility> // For pair
#include <algorithm> // For sort
#include <iostream>
using namespace std;
int main() {
  pair<int,int> a[3]; // Array of integer pairs
  a[0] = make_pair(170,60);
  a[1] = make_pair(180,90);
  a[2] = make_pair(170,65);

  for (int i=0; i<3; ++i) // Display from a[0] to a[2]
    cout << a[i].first << '␣' << a[i].second << endl;
  // Should display:
  // 170 60
  // 180 90
  // 170 65

  sort(a, a+3); // Sort from a[0] to a[2]

  for (int i=0; i<3; ++i) // Display from a[0] to a[2]
    cout << a[i].first << '␣' << a[i].second << endl;
  // Should display:
  // 170 60
  // 170 65
  // 180 90
}
\end{verbatim}
```

Python3

```
\begin{verbatim}
a = [[3,5],[2,9],[3,6]]
print(a) # [[3, 5], [2, 9], [3, 6]]
a.sort()
print(a) # [[2, 9], [3, 5], [3, 6]]
\end{verbatim}
```

Tuples of Three or More Elements

In C++11, the `tuple` type is available to represent tuples of three or more elements.[2] In earlier versions of C++, `pair` was nested, such as `pair<int,pair<int,int> >`, but now it can be written more concisely.

C++11

```
\begin{verbatim}
#include <tuple>
#include <iostream>
#include <algorithm>
using namespace std;
int main() {
  tuple<int,int,int> a = {3,1,4};
  cout << get<0>(a) << "\n";    // 3
  cout << get<1>(a) << "\n";    // 1
  cout << get<2>(a) << "\n";    // 4
```

[2] http://en.cppreference.com/w/cpp/utility/tuple

```

  tuple<int,int,int> array[] = {{3,1,4}, {1,5,9}, {2,6,5}};
  sort(array, array+3);

  tuple<int,int,int> t = array[0];
  cout << get<0>(t) << "\n";    // 1
  cout << get<1>(t) << "\n";    // 5
  cout << get<2>(t) << "\n";    // 9
}
\end{verbatim}
```

4.1.8 Sorting in Descending Order

The standard `sort` function sorts in ascending order. How can we sort in descending order?

1. Sort in ascending order first, then reverse the order (this is sufficient for now).

C++

```
\begin{verbatim}
int A[5] = {3,5,1,2,4};
int main() {
   sort(A,A+5);
   // In C++11, sort(begin(A),end(A));
   reverse(A,A+5); // Reverses the order of the given range
   ...  // Try outputting A to cout
}
\end{verbatim}
```

Python3

```
\begin{verbatim}
a = [3,5,1,2,4]
a.sort()
a.reverse()  # Reverses the order of a
\end{verbatim}
```

2. Use `reverse_iterator`

C++14

```
\begin{verbatim}
  sort(rbegin(A),rend(A));
\end{verbatim}
```

Python3

```python
\begin{verbatim}
a.sort(reverse=True)
\end{verbatim}
```

3. Pass a comparison function (general purpose)

C++11

```cpp
\begin{verbatim}
int A[5] = {3,5,1,2,4};
int main() {
   sort(begin(A),end(A),[](int p, int q){ return p > q; });
   ...  // Try outputting A[i] to cout
}
\end{verbatim}
```

The explanation of the syntax is omitted, but the part `[](int p, int q){ return p > q; }` is an anonymous function that determines the order of two integers.

Python3

```python
\begin{verbatim}
a.sort(key=lambda e: -e)
\end{verbatim}
```

4.2 Greedy Algorithms

4.2.1 Using in the Best Order

Problem **Country Road** (UTPC 2008)

Along a straight road called Country Road, houses are sparsely built. We want to supply power to all houses using a specified number of generators and power lines. To supply electricity to a house, it must be connected to one of the generators via a power line, and the power line incurs a cost proportional to its length. Find the placement of generators and power lines that minimizes the total length of the power lines.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2104&lang=jp`

Hint: If there is one generator, it is necessary to connect all houses with power lines from end to end. If there are two generators, it is possible to save on power lines by not running a power line between houses at one location, starting from the state where all houses are connected with power lines from end to end. In other words, it is advantageous to save the longest section. If there are three generators, it is possible to save on power lines by not running power lines in the longest and second-longest sections between houses, starting from the state where all houses are connected with power lines from end to end. ...

Confirming the Strategy

Before writing the program, try solving the Sample Input by hand.

Example solution:

C++

```
\begin{verbatim}
int T, N, K, X[100000+10], A[100000+10];
int main() {
    cin >> T; // Read the number of datasets
    for (int t=0; t<T; ++t) {
        ... // Read the number of houses and generators
        for (int i=0; i<N; ++i) cin >> X[i]; // Input the positions of the houses
        for (int i=0; i+1<N; ++i) A[i] = X[i+1]-X[i]; // Distance between houses
        ... // Sort array A
        ... // Output the sum of the first max(0,N-1-(K-1)) elements of array A
    }
}
\end{verbatim}
```

Python3 input example: To read a given line as an array, convert it with `list(...)`

Python3

```
\begin{verbatim}
N,K = map(int,input().strip().split('␣'))
X = list(map(int,input().strip().split('␣'))) # Explicit conversion to a list
\end{verbatim}
```

Common Oversight

The number of sections that can be saved does not increase (Since the hint is written in white text, you can read it by copying and pasting if you want).

The reason why this type of solution is called a greedy algorithm is that, without considering the ”combinations” of sections to save, it greedily decides to save individual sections one by one (arranged in order of length) until a threshold is reached.

Problem **Stripies** (Northeastern Europe 2001)

There is a species where two bodies with masses m_1, m_2 combine to form a body with mass $2\sqrt{m_1 \cdot m_2}$. Given the initial state, find the minimum mass when all bodies are combined.

http://poj.org/problem?id=1862

To output to three decimal places, use `printf("%.3f\n", ret);`.

Problem **Princess’s Marriage** (Mock National Preliminary 2008)

Hire guards effectively to minimize the expected number of attacks during the journey. Guards cost 1 unit of money per unit distance and can be hired freely as long as there is a budget.

The input consists of the number of sections N, the budget M, followed by N pairs of distance and expected number of attacks per unit distance ⟨D,P⟩.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2019&lang=jp

Approach: If you are going to hire guards, it is best to have them protect the most dangerous sections. Choose the most dangerous road and hire guards there as long as there is a budget. If the remaining budget cannot cover all sections of the road, the road is divided into safe sections and dangerous sections. As long as there is a budget remaining, repeat the same process in order for the second most dangerous road, the third most dangerous road, and so on. For the remaining dangerous sections, the sum of the expected values is the answer. In this calculation process, it is convenient to represent the road with pairs of ⟨danger level, length⟩ and sort them in order of danger. (Here, the danger level represents the expected number of attacks while moving a distance of 1).

Example solution (input and sorting):

C++

```
\begin{verbatim}
int N, M;
pair<int,int> PD[10010];
int main() {
  while (cin >> N >> M && N) {
    int d, p;
    for (int i=0; i<N; ++i) {
```

```
   cin >> d >> p;
   PD[i] = make_pair(p, d);
   // PD[i].first is the danger level of road i
   // PD[i].second is the length of road i
  }
  ... // Sort PD in descending order
  // Display PD to check if the sorting was successful
  // If the sorting is successful, calculate the answer next
 }
}
\end{verbatim}
```

Example solution (calculating the answer):

C++

```
\begin{verbatim}
  int S = 0;
  for (int i=0; i<N; ++i)
   S += danger level of road[i] * length of road[i];
  // The answer when the budget is 0 is the current value of S
  for (int i=0; i<N; ++i) {
   if (M <= 0) break;
   int guarded = min(M, length of road[i]); // Section to hire guards
for
   S -= danger level of road[i] * guarded;
   M -= guarded;
  }
  S is the answer
\end{verbatim}
```

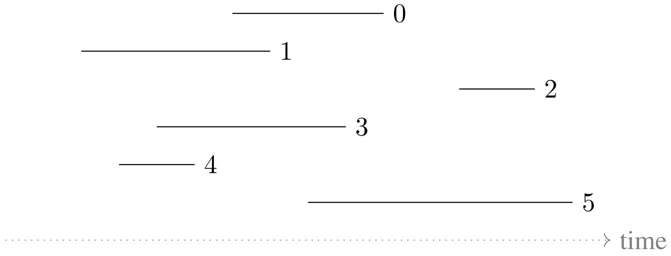
- If the budget is 0, the expected number of attacks S, which is the required answer, is $S = \sum_i P_i \cdot D_i$ for each road i.

- If the budget is M, reduce the expected value from S only for the sections where guards were hired. For example, if guards are hired for the entire section of road j, S can be reduced by $P_j \cdot D_j$. If the remaining budget m is smaller than the length of the road D_j and guards can only be hired for a part of the road, then S can only be reduced by $P_j \cdot m$.

4.2.2 Interval Scheduling

See Chapter 4.1 (pp. 104-108) of ”Algorithm Design” [4].

Problem Overview Suppose there is a single shared resource (e.g., a gymnasium, parking lot) and multiple usage requests (pairs of start and end times). Select a subset of the usage requests such that their usage times do not overlap. What is the maximum number of requests that can be accepted?



Simplification: By removing specifics such as the gymnasium or time, the problem becomes selecting a subset from a set of line segments.

Intuition and Considerations

- Is it sufficient to simply adopt requests from the left? ➔ Not necessarily (there are counterexamples).
- Short line segments are less likely to overlap with others. Is it sufficient to adopt requests starting with the shortest line segments? ➔ Not necessarily (there are counterexamples).

Correct Algorithm

1. Select the *line segment with the leftmost right endpoint* (i.e., the request that finishes earliest), and remove it along with any overlapping line segments.
2. If there are still unselected line segments, return to step 1 and repeat the procedure.

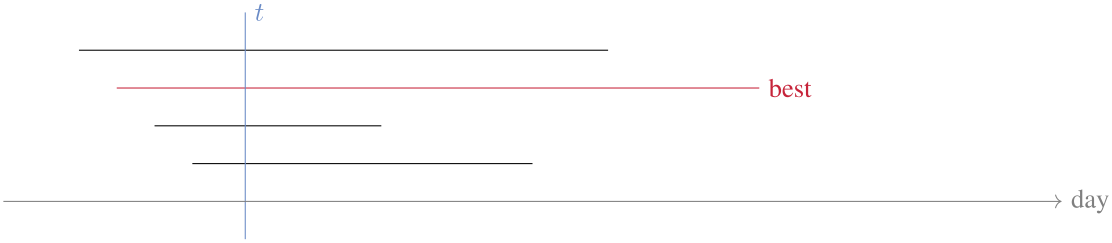
Practice Problems for Handling Intervals The following problems are different from the interval scheduling problem, but they are problems that involve handling intervals.

Problem **Cleaning Shifts** (USACO 2004 December Silver)

We want to determine the cleaning shifts for T consecutive days. Each ”cow” is known to be able to work during a certain interval (including the boundaries) from day X to day Y. When assigning cows to days where there are no assigned shifts, what is the minimum number of cows needed? Output -1 if it is impossible.

`http://poj.org/problem?id=2376`

Hint: Among the cows that can cover the current shift, select the cow that can cover the latest day.



Example Solution (Input/Output)

C++

```
\begin{verbatim}
int /*Total number of cows*/N, /*Number of cleaning days*/T;
pair<int,int> C[25010]; // C[i].first is the start day of the shift, C[i].second is the end day of the shift
int solve() {
  // Calculate the answer based on N, T, and C
}
int main() {
    scanf("%d␣%d", &N, &T);
    for (int i=0; i<N; ++i)
      scanf("%d␣%d", &C[i].first, &C[i].second);
    printf("%d\n", solve());
}
\end{verbatim}
```

Example Solution (Calculation)

C++

```
\begin{verbatim}
int solve() {
    sort(C, C+N); // Sort by the earliest start day of the shift
    int /*Cow number*/i = 0, /*Start day of the next shift*/t = 1, /*Number of cows
    while (i<N && t<=T) {
        int best = 0; // End day of the next cow to be hired
        while (i<N && C[i].first <= t) {
            ... // If the end day of cow i is later than best, update best
            ++i;
        }
        if (best < t) return -1; // If there are no cows available
        t = best+1;
        ++c;
    }
    return (t>T) ? c : -1; // Check if cleaning is finished until T
}
\end{verbatim}
```

Strategy: Assume that cleaning is finished up to day $t-1$. If there are no cows that can clean during the interval including day t, there is no solution (output -1). If multiple cows can cover day t, it is best to hire the cow that can cover the longest interval (*). If that cow covers up to day s, then set $t = s + 1$ and continue hiring cows until the entire interval is covered.

There may be other cows that can lead to an optimal solution, but it can be proven that the number of cows hired cannot be reduced compared to the strategy (*).

Problem **Radar Installation** (Beijing 2002)

Place radars so that all islands are visible.
Note: There seems to be an island at y=0.
http://acm.pku.edu.cn/JudgeOnline/problem?id=1328

Hints:

- Suppose you have listed the intervals where a radar can observe each island. Consider the strategy of placing a radar at the location that covers the most islands that are not yet covered by a radar, and repeating this process. Show an example of an island arrangement where this strategy requires more radars than the optimal arrangement.
- Suppose you place radars from left to right. Select the island that starts being visible at the leftmost position (among those not yet covered by a radar), and place a radar at the left end of the interval where that island is visible. Show an example of an island arrangement where this strategy requires more radars than the optimal arrangement.
- Suppose you place radars from left to right. Select the island that starts being visible at the leftmost position (among those not yet covered by a radar), and place a radar at the right end of the interval where that island is visible. Show an example of an island arrangement where this strategy requires more radars than the optimal arrangement.
- Show the correct strategy A for placing radars from left to right, and prove that no other arrangement can have fewer radars than the arrangement by strategy A.

Common Bug

It is necessary to consider exceptions such as islands that are too far away. Since multiple test cases are handled, even if there are islands that are too far away, the input should read all islands (otherwise, the beginning of the next test case will be misaligned).

4.2.3 Various Problems

Problem **Make Purse Light** (Mock National Preliminary 2005)

Lighten the contents of your wallet.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2007&lang=jp`

Problem **Fox and Card Game** (Codeforces 388 C)

The first player takes cards from the top of the pile, and the second player takes cards from the bottom. Find the scores of each player when they both play optimally.

`http://codeforces.com/problemset/problem/388/C`

Problem **Shopping** (Asian Games 2014)

Find the minimum travel distance required to pass through multiple stores arranged on a straight line from left to right while satisfying certain constraints.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1347&lang=jp`

Problem **Dinner⋆** (Summer Camp 2014)

Maximize happiness over N days by combining cafeteria meals and self-prepared meals.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2642&lang=jp`

Problem **Ploughing⋆⋆** (13th Polish Olympiad in Informatics)

Divide a field by repeatedly cutting it vertically (1 column wide) or horizontally (1 row). Ensure that the sum of numbers in each section is less than or equal to K. Find the minimum number of divisions that satisfy the condition.

(Problem continued)

`https://szkopul.edu.pl/problemset/problem/6YiP6JA5U15hY94pLwuHoYPg/site/`

Chapter 5

Dynamic Programming (1)

Overview

Some problems that are difficult to solve by exhaustively checking all possibilities can be easily solved by appropriately organizing them, such as by pre-solving smaller problems and storing the solutions in a table. Let's formulate equations that represent the relationships between larger and smaller problems and implement them in a program. Dynamic programming is a method that achieves efficient computation by utilizing the optimal substructure property of problems.

Statue of Fibonacci Saint (Camposanto, Pisa)

5.1 Counting Numbers

Example **Fibonacci Number** (AOJ)

Find the Nth Fibonacci number.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_10_A

For simplicity, let F_i denote the i-th Fibonacci number. It is defined as follows (it is often defined with $F_0 = 0$, but it is not essential, so we follow the problem statement):

$$F_i = \begin{cases} 1 & i = 0 \\ 1 & i = 1 \\ F_{i-1} + F_{i-2} & \text{(otherwise)} \end{cases} \tag{5.1}$$

A straightforward recursive definition based on the above would be:

Python3

```
def fib(n):
    print("fib",n) # Display the argument when the function is called (remove after confirming operation)
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2)+fib(n-1)
```

However, this method involves a lot of redundant calculations and is not efficient. It becomes rapidly slower as n increases. As an example, here are execution time measurements using Python's `timeit` module.

Python3

```
import timeit
print(timeit.timeit("fib(10)", globals=globals(), number=10))
print(timeit.timeit("fib(20)", globals=globals(), number=10))
print(timeit.timeit("fib(30)", globals=globals(), number=10))
```

```
\$ python3 fib.py
0.00014154397649690510
0.01775405099033378
2.1516372300102375
```

The more straightforward method we will primarily use is to calculate the Fibonacci numbers in order from the smallest. From equation (5.1), we can see that to find F_i, we only need the values from F_0 to F_{i-1}, and with those, it can be calculated with one addition. Therefore, if we calculate in order from F_0, each F_i can be found with one addition.

C++

```
int F[100]; // As needed
int main() {
  F[0] = 1;
  F[1] = 1;
  for (int i=2; i<45; ++i) { // As needed
    F[i] = F[i-2]+F[i-1];
  }
}
```

The problems we will deal with in this chapter are those where the answer to the problem we want to solve (e.g., F_{100}) can be efficiently calculated from the answers to subproblems ($F_n,\ n < 100$). In this case, although only the N-th data is directly needed for the answer, an approach that calculates all the data *up to* the N-th is effective, even if it seems like a detour. The total number of basic operations required (hereinafter simply referred to as the computational complexity) is $O(N)$. Note that for $O(\log N)$ calculation methods using memoization or repeated squaring[1], please refer to Chapter 12.

The goal of this section is to be able to write a program to find the solution from a recurrence relation like equation (5.1), which represents the relationship between the solutions of subproblems.

Example **Kannondou** (PC Koshien 2007)

A person can climb 1, 2, or 3 steps at a time. Output the number of ways to climb n (< 30) steps, formatted appropriately.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=0168&lang=jp

Represent the number of ways to climb to each step using an array. Recurrence relation: As a subproblem, let A_i be the number of ways to reach the i-th step from the bottom. Before climbing, $A_0 = 1$, and what we ultimately want to know is A_n.

$$A_i = \begin{cases} 1 & i = 0 \\ A_{i-1} & i = 1 \\ A_{i-1} + A_{i-2} & i = 2 \\ A_{i-1} + A_{i-2} + A_{i-3} & \text{(otherwise)} \end{cases} \tag{5.2}$$

Similar to the calculation of Fibonacci numbers, the computational complexity for finding the answer for N steps is $O(N)$.

Example solution (number calculation):

[1] For simplicity, this material treats the cost of integer operations as constant. However, when using multi-precision integers, note that operations on large numbers become slower.

C++

```cpp
#include <algorithm>
#include <iostream>
using namespace std;
int A[128], N;
int main() {
    A[0] = 1;
    for (int i=1; i<=32; ++i) {
        A[i] = A[i-1];
        if (...) A[i] += A[i-2];
        if (...) A[i] += A[i-3];
    }
    // Try outputting A[.]  appropriately
}
```

Python3

```python
A = [0 for _ in range(32)]
A[0] = 1
for i in range(1,32):
    A[i] = A[i-1]
    if i > 1:
        A[i] += A[i-2]
        if i > 2:
            A[i] += A[i-3]
```

Beware of Out-of-Bounds Array Access

In C and C++, you must not refer to or write to out-of-bounds array locations, for example, `A[-1]` (it may cause an immediate runtime error, or it may not, or it may cause even more trouble). Typically, if you write a statement that accesses `A[i-2]`, the programmer must ensure that it is not executed when `i`$\leq$ 1.

Example solution (input/output):

C++

```cpp
    while (cin >> N && N)
        cout << ((A[N]+..)/10+...)/365 << endl;
```

To round up an integer, add the divisor - 1 before dividing. In Python3, use `//` for integer division. Note that using floating-point numbers and `ceil` can cause calculation errors.

Python3

```python
while True:
    n = int(input())
    if n == 0:
        break
    print(((A[n]+9)//10+364)//365)
```

Reference if the Answer is Incorrect

For 15 steps, there are 5768 ways, which takes 2 years.

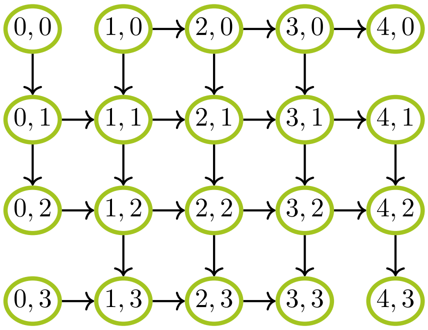
Problem **Heiankyo Walking** (UTPC2009)

Count the number of paths to reach the goal from the start in a city on a grid (only walking in the direction of approaching the goal), where paths with catnip are impassable.

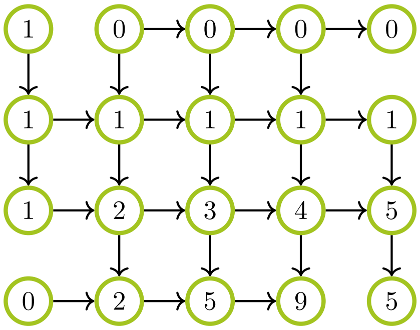
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2186&lang=jp

Note that if there is no catnip, it can be calculated immediately from the concept of combinations (when to use the vertical roads out of the total number of vertical and horizontal roads to walk to the destination).

If there is catnip, a natural solution is to count the number of reachable paths at each intersection. The situation for the third sample input is as shown in the figure below.

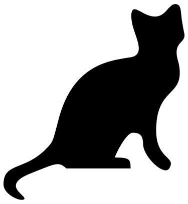


Coordinates of intersections and passable roads (movable right or down)



Number of paths reachable at each intersectio

As a subproblem, consider the number of paths reachable at a certain intersection (x, y), and denote it as $T_{x,y}$. This value can be calculated as the sum of the values of the intersections that can be reached in one step (usually left and above), if those values are determined.



$$T_{x,y} = \begin{cases} 0 & (x,y) \text{ is out of bounds} \\ 1 & (x,y) = (0,0) \\ 0 & \text{Catnip both above and to the left} \\ T_{x-1,y} & \text{Catnip only above} \\ T_{x,y-1} & \text{Catnip only to the left} \\ T_{x-1,y} + T_{x,y-1} & \text{No catnip above or to the left} \end{cases}$$

Since the catnip input is given in a somewhat redundant format, it is convenient to preprocess it as follows and store it in an array that indicates impassability.

- Same x-coordinate – (x1, max(y1,y2)) is not movable from above

- Same y-coordinate – ((max(x1,x2), y1) is not movable from the left

For example, manage whether movement is possible from above and from the left at a certain position (x,y) using two two-dimensional arrays, `Vert[x][y]` and `Horiz[x][y]`. If movement from above to a certain point (x,y) is possible, then `Vert[x][y]==true`, otherwise `Vert[x][y]==false`. Similarly, set each element of `Horiz[x][y]` based on whether movement from the left is possible. Initialize each element to `true` beforehand, and rewrite to `false` if there is catnip. (If representing *impossibility* of movement, the correspondence between `true` and `false` is reversed. Either is fine as long as it is not confusing.)

Recommended Program Creation Procedure

Display the number of paths reachable at each intersection as shown in the right side of the previous page's figure, and check if it matches your manual calculation.

Hint

Pay atter
fact that

Sentinel Method: Hint for a Clear Program

Similar to the "Kannondou" example, it is necessary to prevent accessing `T[x-1][y]` when `x==0` and to prevent accessing `T[x][y-1]` when `y==0`. Although it can be written with `if` statements, it can be written more concisely by assuming that there is virtual catnip at the top and left edges and setting `Vert[x][0]` and `Horiz[0][y]`.

5.2 Finding and Restoring Optimal Paths

Problem **Spiderman** (Tehran 2003 Preliminary)

Complete a training menu that involves repeatedly climbing or descending by specified heights $H[i]$ and return to the ground. Minimize the maximum height required during the menu.
`http://poj.org/problem?id=2397`

In this problem, we need the minimum value, not the total value.

Let H_i be the height change at the i-th move (i starts from 0), and let $T_i[h]$ be the minimum cost (maximum height in the path) required to be at height h at the i-th building. Initialize these values with $T_0[0] = 0$ (since we start on the ground) and the rest with ∞. Then, sequentially calculate T_i and T_{i+1} from the relationship with the next building.

$$T_{i+1}[h] = \min \begin{cases} \max(T_i[h - H_i], h) & \cdots \text{If climbing from the } i\text{-th building}(h \geq H_i) \\ T_i[h + H_i] & \cdots \text{If descending from the } i\text{-th building} \end{cases}$$

Let M be the goal location. Since we are on the ground at the goal, $T_M[0]$ gives the minimum value.

Since we cannot go below the ground, we only consider non-negative heights. Also, since we cannot descend to the goal if we climb too high, we only need to consider up to a certain height.

Furthermore, this problem requires not only the minimum value but also the **path** that gives the minimum value. This can be found in one of two ways:

1. After finding the minimum value $T_M[0]$, trace back from the goal to the start. Whether we climbed or descended from the previous building can be determined from the relationship between T_i and T_{i+1} (if both have the same value, either is fine).
2. When updating $T_{i+1}[h]$, record whether we climbed or descended in $U_{i+1}[h]$. After finding $T_M[0]$, trace back from $U_M[0]$ to $U_{M-1}[H_0]$.

5.3 Various Dynamic Programming Techniques

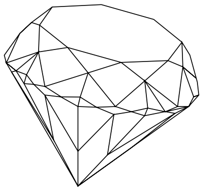
Knapsack Problem Given several treasures with values and weights, we want to select the most valuable items without exceeding the knapsack's capacity (the total weight that can be carried). If the product of the number of item types and the possible numerical values of the weights is within a certain limit, it can be solved using dynamic programming. Note that if the quantity can be freely adjusted, like a liquid – which is a completely different problem –, it is sufficient to greedily select items with the highest value per weight (dynamic programming is not needed).

Problem **Combinatorial - 0-1 Knapsack Problem** (AOJ)

In the 0-1 knapsack problem, where each item can be selected at most once, find the maximum total value that satisfies the constraints.

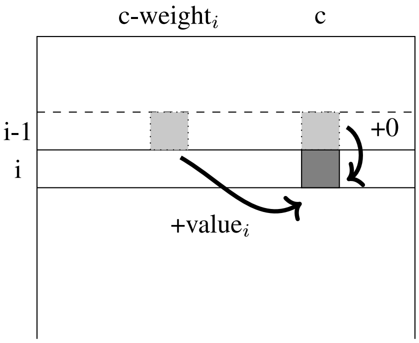
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=DPL_1_B&lang=jp

If the capacity is an integer and relatively small (can be stored in an array), the following method is effective. First, assign a serial number to each item (the order does not matter). Then, as a subproblem, consider $V_{i,c}$, the maximum total value that can be carried when there are only items up to item i and the knapsack's weight limit is c.



$$V_{i,c} = \begin{cases} 0 & i \leq 0 \text{ or } c \leq 0 \\ V_{i-1,c} & c - \text{weight}_i < 0 \\ \max(\text{value}_i + V_{i-1,c-\text{weight}_i}, V_{i-1,c}) & \end{cases}$$

$V_{i,c}$ is the maximum value considering whether to select the i-th item or not, but in either case, it can be calculated using $V_{i-1,*}$. Therefore, the entire calculation can be performed efficiently using a two-dimensional array. See Reference$^{\text{Strategy}}$[2, pp. 416–].



Problem **Combinatorial - Knapsack Problem** (AOJ)

In the knapsack problem where each item can be selected any number of times, find the maximum total value that satisfies the constraints.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=DPL_1_C&lang=jp

The approach is almost the same, but the recurrence relation changes slightly. This knapsack problem without quantity restrictions can be solved with the same computational complexity order $O(NW)$ as the 1-item-limited knapsack problem. If you plan to increase the `for` loop related to weight by one, making it a triple loop, it will become $O(NW^2)$. Consider how to improve it.

As a similar problem, there is the knapsack problem with quantity restrictions, where each item has a given quantity constraint (bound). By applying the concept of sliding minimum values, a solution can be obtained more efficiently (than solving it as a 0-1 knapsack with increased items) (see also Reference[3, p. 302]).

Longest Common Subsequence and Edit Distance A subsequence is a sequence obtained by extracting some elements from a data sequence such as a string (or a sequence obtained by deleting some elements and packing the remaining elements). A common subsequence is a sequence that is a subsequence of both of two data sequences. The longest sequence among the common subsequences is called the longest common subsequence. There may be multiple longest common subsequences.

Problem **Dynamic Programming - Longest Common Subsequence** (AOJ)

Find the length of the longest common subsequence.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_10_C&lang=jp

Approach: Let X_i represent the subsequence obtained by extracting the first i characters of string X. X_0 is an empty string. The longest common subsequence of string X and string Y can be found using $L_{i,j}$, the length of the longest common subsequence of X_i and Y_j, which is a subproblem.

$$L_{i,j} = \begin{cases} 0 & i \leq 0 \text{ or } j \leq 0 \\ 1 + L_{i-1,j-1} & \text{The } i\text{-th character of X and the } j\text{-th character of Y are the same} \\ \max(L_{i,j-1}, L_{i-1,j}) & \text{otherwise} \end{cases}$$

This calculation can be performed efficiently using a two-dimensional array. See the ”Pattern Recognition” chapter of information science or Reference$^{\text{Strategy}}$[2, pp. 253–].

Note: It is difficult to fit this problem within the time limit in Python. If you get TLE, it is good to download the data and verify the answer manually.

Problem **Combinatorial - Edit Distance (Levenshtein Distance)** (AOJ)

Find the edit distance between two strings.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=DPL_1_E&lang=jp

Longest Increasing Subsequence The longest increasing subsequence is a well-known problem in dynamic programming, and it can be solved in $O(N \log N)$ by combining `vector` and binary search (see Section 6.1). See the explanation at the end of the problem or Reference$^{\text{Strategy}}$[2, pp. 421–].

Problem **Combinatorial - Longest Increasing Subsequence** (AOJ)

Find the Longest Increasing Subsequence.
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=DPL_1_D&lang=jp`

5.4 Exercises

Problem **Minimal Backgammon** (Asian Regional 2007)

Find the probability of reaching the goal in a Sugoroku game on a straight line.
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1277&lang=jp`

Problem **Coin Changing Problem** (AOJ)

Find the minimum number of coins needed to pay exactly.
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=DPL_1_A&lang=jp`

Note: Japanese coins have the property that the value of a larger coin is divisible by the value of a smaller coin, so it is better to use larger coins as much as possible. That is, if the payment amount exceeds 500 yen, it is better to use a 500 yen coin. On the other hand, this problem also deals with situations where this is not the case. When paying 200 yen with a coin system of 150 yen, 100 yen, and 1 yen coins, using a 150 yen coin does not minimize the number of coins. See Reference$^{\text{Strategy}}$[2, pp. 412–].

Problem **Eleven Lover**$\star$ (Mock Regional 2009)

Count the multiples of 11 in a sequence of digits.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2182&lang=jp

Example approach: Look at the digits one by one from the left. Focusing on the i-th digit, let $A_{i,j}$ be the number of numbers ending at the i-th digit that have a remainder of $0 \leq j < 11$ when divided by 11. Calculate $A_{i,\cdot}$ from $A_{i-1,\cdot}$ and the value of the i-th digit.

Problem **Restore Calculation**$\star$ (Asian Regional 2013)

Given two integers and their sum, which may contain ?, find the number of ways to fill in the ?'s such that the operation is consistent.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2566&lang=jp

Problem **Magic Slayer**$\star$ (Summer Camp 2009)

Defeat all enemies by skillfully using single-target and area-of-effect magic. Each magic has a specified MP cost and damage, in addition to being single-target or area-of-effect. The enemies are given by their HP values, which indicate how much damage is needed to defeat them.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2156&lang=jp

Explanation: http://acm-icpc.aitea.net/index.php?plugin=attach&refer=2009%2FPractice%2F%B2%C6%B9%E7%BD%C9%2F%B9%D6%C9%BE&openfile=2b.pdf

You can ignore the names of the magic spells.

Consider the case of a single enemy with only single-target magic. Let A_d be the minimum MP required to deal a total damage of d. Then,

$$A_d = \begin{cases} 0 & d \leq 0 \\ \min_k(A_{d-\text{Damage}_k} + \text{MP}_k) & d > 0 \text{ using magic } k \text{ last} \end{cases} \tag{5.3}$$

When defeating N enemies with only single-target magic, it is sufficient to sum the required MP for each enemy's HP.

When area-of-effect magic is available, calculate the minimum MP required to deal a total damage of d to all enemies in the same way, and combine them.

Problem **Mushrooms**$\star$ (Algorithmic Engagements 2010)

Collect mushrooms by walking on a straight line.
https://szkopul.edu.pl/problemset/problem/KVIxa_im2wGYJX99NB31p_nC/site/

Recommended to use scanf.

Problem **Rotating Blackout Plan**$\star$ (National Preliminary 2011)

Divide the intervals skillfully.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1176&lang=jp

A type that manages values for rectangles.

Problem **Quest of Merchant**$\star$ (Summer Camp 2011)

Maximize profit by combining various actions (see problem statement).
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2296&lang=jp

Problem **My friends are small** (Summer Camp 2010)

There are N items. How many ways are there to choose a maximal subset of items whose total weight is less than or equal to W? $N \leq 200, W \leq 10\,000$ http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2333&lang=jp

Maximal means that adding any unselected item would exceed W.

Approach: Among the unselected items, consider the lightest one and branch based on it. If we calculate the number of ways to choose items with a total weight of w using a knapsack problem in descending order of weight, we can obtain the solution from the information in the solving process.

Explanation: https://drive.google.com/file/d/1WC7Y2Ni-8elttUgorfbix9tO1fvYN3g3/view

Problem **Hakone⋆⋆** (Summer Camp 2012)

From the changes in rankings (e.g., this team moved up in the rankings and passed in 5th place), find the number of possible rankings at the previous relay point (for Hakone Ekiden).

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2439&lang=jp

Problem **Barricades** (Algorithmic Engagements 2007)

Problem: Block off some roads with barricades to make a connected set of k cities inaccessible from other cities.

https://szkopul.edu.pl/problemset/problem/1sX3vqLjiqkpxBNI-sh8UC2m/site/

Problem **Army Training⋆⋆** (Algorithmic Engagements 2010)

Given up to 1000 points on a plane, connect them appropriately to form a simple polygon, and count the number of points contained within the area.

https://szkopul.edu.pl/problemset/problem/nXF1qOIv5S88utFPI2V_0gt3/site/

Chapter 6

Divide and Conquer

Overview

Divide and conquer is a technique that recursively decomposes a problem into smaller subproblems, solves each of them, and then sequentially combines them to obtain the overall solution. It is similar to dynamic programming in that it deals with problems smaller than the original problem, but divide and conquer handles cases where the divided subproblems do not overlap. For example, binary search is a divide and conquer technique, but the calculation of Fibonacci numbers is not usually called divide and conquer.
This chapter deals with recursion. If you are not familiar with recursion, please refer to Appendix B.2 as needed.

In this chapter, be especially careful because i (`i`) and j (`j`), and 1 (`1`) and l (`l`) are particularly confusing.

6.1 Binary Search

Let's look at binary search as an example of a case where it is sufficient to divide a problem in half and deal with only one side. For example, consider the case of determining whether an element exists in a data sequence that is ordered, such as a dictionary or a roster. For instance, when searching for the surname "Moore" in a roster with a total of 1024 pages, start from the middle page, page 512. If that page is after "M", search the front. If it is before "M", search the back. In either case, the search range can be narrowed down to half.

This search method is faster than searching the roster page by page from the first page to the last. If the number of pages in the roster is n, there is a difference between the number of pages to check, $O(n)$ and $O(\log(n))$.

Note that in merge sort and `inversion count`, which will be introduced in the next section, it is necessary to process both regions after division.

6.1.1 Searching for Values in a Data Sequence

The `binary_search` function, included in the C++ standard library, uses binary search to determine whether an element exists in a sorted array.

C++

```
\begin{verbatim}
#include <iostream>
#include <algorithm>

sort(S, S+L); // Preparation:  Sort the array S in ascending order beforehand
...
    if (binary_search(S, S+L, a)) {
       // a is in S
    }
\end{verbatim}
```

If we write the case of searching for `value` from the array `A[]` somewhat formally, it would be as follows:

1. Let the interval to search be $[l, r)$ (to represent a range, left, right, or first, last are often used). When searching from page 0 to page 1023, the initial state is `l=0, r=1024`.
2. If `l+n <= r`, the search range has at most `n` pages left, so search the range `[l,r)` sequentially. (Typically `n==1`, but in practice, it may be set to around `n==10`).
3. Find the middle of the interval: `m=(l+r)/2`.
4. If `value < A[m]`, (we want to search the first half `[l,m)` next), set `r=m`. Otherwise (including the case where `A[m] <= value`, i.e., `A[m]==value`), we want to search the second half `[m,r)`, so set `l=m` and go to step 2. (Here, if `m==l` or `m==r`, it will be an infinite loop. Confirm that this does not happen.)

This is the kind of process that is performed.

C++

```
\begin{verbatim}
bool bsearch(const int array[], int left, int right, int value) {
    // Assumptions:
    // - array is sorted in ascending order,
    // - the half-open interval [left,right) is a valid range
    // Possibilities
    // A. array[left] <= value <= array[right-1] (answer is possible)
    // B. value <= array[left] (false is certain)
    // C. array[right-1] <= value (false is certain)
    while (left + 1 < right) {
        // Loop invariant:  One of A, B, or C that holds at the start of the loop also holds at the end
    int med = (left+right)/2;
    if (array[med] > value) right = med;
    else left = med;
  }
  return left < right && array[left] == value;
}
\end{verbatim}
```

After implementing binary search yourself, you can check its operation with Search II.

There is a concept called *loop invariant* as a tool for thinking about the correctness of programs that include complex loops. If necessary, please refer to Appendix B.1.

Example **Search II** (AOJ)

Given an array of integers S (a list of items to sell) and T (a list of items to buy). Output the number of integers in T that are also included in S.

(Input range) The number of elements in S is at most 100,000, and the number of elements in T is at most 50,000. The number of elements in S and T are each at most 100. The elements included in the arrays are between 0 and 10^7.

(Note) The elements of T are mutually distinct, but the elements of S may have duplicates.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=10031&lang=jp

Pay Attention to the Problem Specifications

The meaning of ”The elements of T are mutually distinct, but the elements of S may have duplicates” is that if `S=[1,1,1], T=1`, then the answer is 1.

6.1.2 Finding the Minimum Value that Satisfies a Constraint

Besides situations where we search for a value in an array, there are problems that can be solved elegantly using the concept of binary search.

Example **Search - Allocation** (AOJ)

Consider loading items of various weights onto trucks in the order they are arranged. Assume the weights of the items are stored in an array `w` in the order they are arranged. Assume that all trucks

(Example continued)

have the same maximum load capacity. What is the minimum maximum load capacity required for each truck to load all the items onto k trucks in order? Find the minimum value of the ”maximum load capacity of the truck”.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_4_D&lang=jp

Hint: Perform a binary search on the maximum load capacity P of the truck to find the minimum value that allows all items to be loaded. Since the value we are looking for, ”the minimum value that allows loading”, is an integer, we can find ”the maximum value that does not allow loading” and add 1, because ”the minimum value that allows loading” = ”the maximum value that does not allow loading” + 1. Represent the range where ”the maximum value that does not allow loading” exists as `[l,h)`, and express the condition for reducing the upper bound `h` as `ok`. The answer we are looking for is `h`.

C++

```
\begin{verbatim}
bool ok(int P) {
  ... // Returns whether the number of trucks needed when loading in order with a maximum load capacity of P is within K.
}
int main() {
    // Read the problem
    int l = 0, h = // A large number;
    while (l+1 < h) {
        // Loop invariant:  Cannot load with l, can load with h
        int m = (l+h)/2;
        if (ok(m)) h = m; else l = m;
    }
    printf("%d\n", h);
}
\end{verbatim}
```

The function `ok` can be created as follows, for example:

- Create variables representing the weight loaded on the current truck (initial value 0) and the number of trucks (initial value 1).
- For each item, if loading it onto the current truck does not exceed the maximum load capacity, load it (increase the weight loaded on the current truck). Otherwise, load it onto a new truck (increase the number of trucks and set the weight loaded on the current truck to that item).

- Finally, compare the number of trucks with K.

Example **Aggressive Cows⋆** (USACO 2005 February Gold)

There are N stalls (individual rooms) for cows on a straight line. We want to place C cows as far apart from each other as possible.

(Please use scanf instead of cin)

http://poj.org/problem?id=2456

This time, since we want the maximum value that satisfies the condition, we express the condition for increasing the lower bound `l` as a function `ok`. The answer we are looking for is `l`. If we sort the candidate stalls in ascending order beforehand, the `ok` function can be implemented as follows:

- Place a cow in the leftmost stall (fail if there are no stalls).
- Remove stalls within a distance of M from consideration.

Repeat these steps for the number of cows, and if all cows can be placed, it can be judged as a success.

Example **Water Tank** (Mock Regional Preliminary 2009)

Given the capacity of a tank and the amount of water used per hour, find the minimum required water supply rate. (The amount that does not lead to a gradual decline even after multiple cycles is required).

(Note that there is another problem with the same name)

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2180&lang=jp

Assume a certain water supply rate, and let `ok` be a function that determines whether it is possible to live with that rate. Represent the interval containing the required rate as $[\mathrm{lo},\mathrm{hi}]$, and narrow the range in half depending on whether it is possible to live or not. When the interval becomes sufficiently narrow, terminate and output the result. The concept and effect are almost the same as binary search, but when dealing with continuous intervals, it is often called bisection.

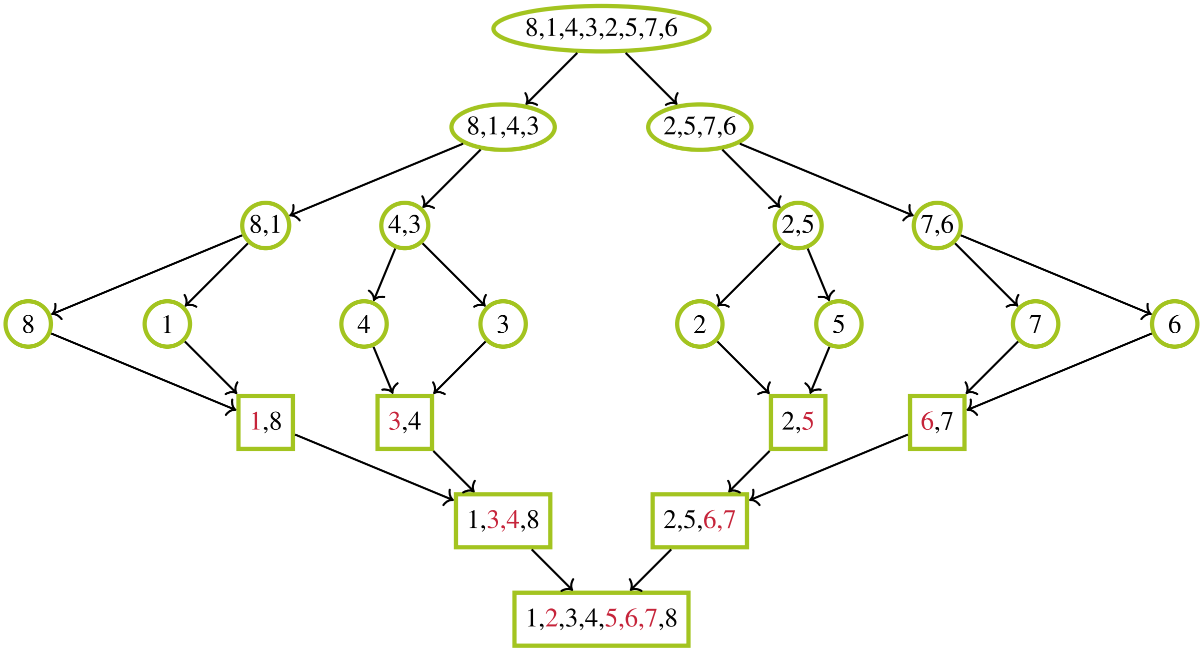
C++

```
\begin{verbatim}
  double lo = sum/86400, hi = 1e6; // 1e6 = 10^6
  while (hi-lo>1e-6) {
      double m = (hi+lo)/2.0;
      if (ok(m)) hi = m;
      else lo = m;
  }
  printf("%.10f\n", hi);
\end{verbatim}
```

The principle is simple, but in the implementation of the function `ok`, it is necessary to be careful in handling cases such as not being able to store water beyond the maximum capacity of the tank. Therefore, the problem of inversion count in the next section might be easier.

6.2 Merge Sort

As an example of divide and conquer, we will discuss a sorting method called Merge Sort. This method divides a given array in half and sorts the elements when assembling it back to its original size. The figure shows the flow of processing when sorting the array 8, 1, 4, 3, 2, 5, 7, 6. The division occurs in the upper half, and the sorting occurs in the lower half. The red numbers indicate elements that came from the right side during merging. The specific source code is almost the same as the `merge_and_count` introduced next, so it is omitted. If the number of elements in the array is N, there are $O(\log N)$ rows, and the calculation required for each row is $O(N)$, so the overall computational complexity is $O(N \log N)$.



6.2.1 Inversion Count

We want to count the number of pairs of elements in an array such that `A[i] > A[j] (i<j)`. For example, in the array `[3 1 2]`, there are two pairs with reversed order: (3,1) and (3,2). If we naively write the following code, it will take time proportional to the square of the number of elements N, i.e., $O(N^2)$.

C++

```cpp
int N, A[128];
int solve() {
    int sum = 0;
    for (int i=0; i<N; i++)
        for (int j=i+1; j<N; j++)
            if (A[i] > A[j]) ++sum;
    return sum;
}
```

By applying merge sort and counting while dividing in half, it can be found in $O(N \log N)$.

C++

```cpp
int N, A[lots]; // A is the original array
int W[lots]; // W is a work array
int merge\_and\_count(int l, int r) { // range [l,r)
    if (l+1 >= r) return 0; // empty
    if (l+2 == r) { // [l,r) == [l,l+1] only 2 elements
        if (A[l] <= A[l+1]) return 0;  // no inversion
        std::swap(A[l], A[l+1]);
        return 1; // one inversion
    }
    int m = (l+r)/2; // [l,r) == [l,m) + [m,r)
    int cl = merge\_and\_count(l, m); // recursively count the left half
    int cr = merge\_and\_count(m, r); // recursively count the right half
    int c = 0; // number of inversions when merging left and right
    int i=l, j=m; // i moves in [l,m), j moves in [m,r),
    int k=l;      // write smaller elements to W[k]
    while (i<m && j<r) { // compare A[i] and A[j] while moving
        if (A[i] <= A[j]) W[k++] = A[i++]; // left half is smaller, no inversion
        else {
            W[k++] = A[j++];
            c += m - i; // left half is larger, skip unprocessed elements
in the left half
        }
    }
    while (i<m) W[k++] = A[i++]; // if left half remains
    while (j<r) W[k++] = A[j++]; // if right half remains
    assert(k == r);
    std::copy(W+l, W+r, A+l);
    return cl + cr + c;
}
```

6.2.2 Exercises

Problem **Recursion / Divide and Conquer - The Number of Inversions** (AOJ)

Find the number of pairs in a given sequence where the order of magnitude is reversed.
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_5_D&lang=jp`

Problem **Ultra-QuickSort** (Waterloo local 2005.02.05)

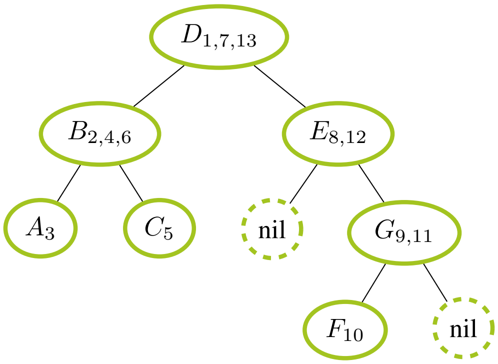
Similar problem: Use `long long` because the numbers are large.
`http://poj.org/problem?id=2299`

Problem **Japan**⋆ (Southeastern Europe 2006)

Roads run east-west on a rectangular island. Find the number of intersections.
`http://poj.org/problem?id=3067`

Example solution: If you sort by (East, West) pairs, it becomes the same problem as the previous two. This problem can also be solved by managing cumulative sums.

6.3 Tree Traversal



Example 1: Numbers represent the order of visitation

Starting from the root, consider traversing each vertex once while following the edges using the following recursive procedure (depth-first search, Chapter **??**):

1. If a left child exists and has not been visited, visit the left child.
2. (Otherwise) If a right child exists and has not been visited, visit the right child.
3. (Otherwise) If there is a parent, return to the parent.
4. If none of the above, (since we have returned to the root) terminate.

For the tree in ”Example 1” in the figure, the path would be DBABCBDEGFGED. As can be seen from the example, vertices with children are visited multiple times (specifically, the degree of the vertex).

Based on this visitation order, methods for processing each vertex only once include preorder (self, left, right), inorder (left, self, right), and postorder (left, right, self). Each method has a different priority order between children and self.

	DBABCBDEGFGED
preorder	DBA C EGF
inorder	ABC DE FG
postorder	A CB FGED

Problem **Tree - Reconstruction of a Tree** (AOJ)

For a certain binary tree, the list of vertices output in preorder (root, left, right) and the list output in inorder (left, root, right) are given. (Restore the original tree) and output its vertices in postorder (left, right, root). Note that in the original tree, different vertices have different labels.

(Problem continued)

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_7_D&lang=jp

Problem **Tree Recovery**⋆ (Ulm Local 1997)

Same as above (except that multiple cases are given and the format is slightly different).
http://poj.org/problem?id=2255

Hint:

- Let the root be A, the left subtree be L (which may be null), and the right subtree be R. In preorder, they are arranged as AL'R'. (L' and R' are their respective preorder notations). In inorder, they are arranged as L"AR" (L" and R" are their respective inorder notations).
- The root A can be immediately identified from the beginning of the preorder notation.
- The L and R parts can be identified by searching for A in the inorder notation.
- Note that the number of characters is the same for L' and L", and for R' and R".
- The problem of restoring the left tree from L' and L" is a smaller version of the original problem.

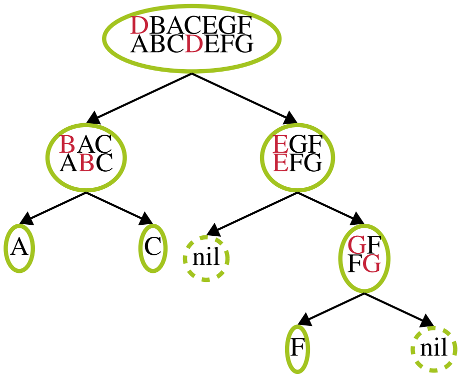
Example solution:

C++

```
\begin{verbatim}
#include <iostream>
#include <string>
using namespace std;
string preorder, inorder;
// For the range [fp,lp) of preorder and the range [fi,li) of inorder,
// display the tree in postorder
void recover(int fp, int lp, int fi, int li) {
    int root;
    // Find the root such that preorder[fp] == inorder[root]
    if (/* If the left side exists */ fp+1 < lp && fi < li)
      recover(fp+1, fp+(root-fi)+1, fi, root); // Display the left side
    if (/* If the right side exists */ fp+(root-fi)+1 < lp && root+1 < li)
      recover(fp+(root-fi)+1, lp, root+1, li); // Display the right side
```

```
        cout << inorder[root]; // Display the root
}
int main() {
    while (cin >> preorder >> inorder) {
        recover(0, preorder.size(), 0, inorder.size());
        cout << endl;
    }
}
\end{verbatim}
```

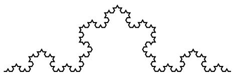
The call relationship can be visualized as follows. In the figure, the red characters represent the root (in the subtree at that point). The strings inside the vertices are the preorder notation (top, range of fp and lp) and the inorder notation (bottom, range of fi and li) for the subtree below that vertex.



6.4 Space-Filling Curves

Problem **Recursion / Divide and Conquer - Koch Curve** (AOJ)

Compute the vertices of the Koch curve.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_5_C&lang=jp



Hint: When a base line is drawn from $(x0, y0)$ to $(x0 + dx, y0 + dy)$, to create a bulge on the left side, extend an appropriate distance from the midpoint $(x0 + dx/2, y0 + dy/2)$ in the $(-dy, dx)$ direction.

There are two approaches to designing a recursive function to draw the Koch curve: one is a design that displays the curve, and the other is a design that returns a sequence of points.

If we design a function `koch(x0,y0,x1,y1,n)` to ”display the points from coordinates (x0,y0) to (x1,y1) (the endpoints include (x0,y0) but not (x1,y1))”, the structure would be as follows:

Python3

```
import math
def show_vertex(x,y):
    print("
def␣koch(x0,y0,␣x1,y1,␣n):
␣␣␣␣if␣n␣==␣0:
␣␣␣␣␣␣␣␣show_vertex(x0,y0)
␣␣␣␣else:
␣␣␣␣␣␣␣␣...␣#␣Calculate␣intermediate␣points␣c1,␣c2,␣c3
␣␣␣␣␣␣␣␣koch(x0,y0,␣c1[0],c1[1],␣n-1)
␣␣␣␣␣␣␣␣koch(c1[0],c1[1],␣c2[0],c2[1],␣n-1)
␣␣␣␣␣␣␣␣koch(c2[0],c2[1],␣c3[0],c3[1],␣n-1)
␣␣␣␣␣␣␣␣koch(c3[0],c3[1],␣x1,y1,␣n-1)

N␣=␣int(input())
koch(0,0,100,0,N)
show_vertex(100,0)
```

In the case of returning a sequence of points, it can be written as follows (display all at once after the entire calculation is finished):

Python3

```
def koch(x0,y0, x1,y1, n):
    if n == 0:
        return [(x0,y0)]
    else:
        ... # Calculate intermediate points c1, c2, c3
        return koch(x0,y0, c1[0],c1[1], n-1) \
            + koch(c1[0],c1[1], c2[0],c2[1], n-1) \
            + koch(c2[0],c2[1], c3[0],c3[1], n-1) \
            + koch(c3[0],c3[1], x1,y1, n-1)

def show_vertex(x,y):
    print("

N␣=␣int(input())
V␣=␣koch(0,0,100,0,N)+[[100,0]]
for␣x,y␣in␣V:
␣␣␣␣show_vertex(x,y)
```

In C++, since the implementation of the latter design is cumbersome, the former design (displaying within the recursive function) is simpler.

Visualizing Point Sequences

When the answer is incorrect or you are struggling with the approach, it is good to visualize the point sequence. In an environment where `gnuplot` is available, it can be easily visualized using the following procedure. (1) Start gnuplot in the terminal. (2) At the gnuplot prompt, type `plot 'xy.txt' with lines`. Assume that the coordinates output by the program are saved in “xy.txt”.

```
\$ gnuplot
gnuplot> plot 'xy.txt' with lines
```

Also, in the case of Python, `matplotlib` is convenient to use.

Python3

```python
import matplotlib.pyplot as plt
import numpy as np
def plot_koch(a):
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    a_ = np.array(a)
    xlist = a_[:, 0]
    ylist = a_[:, 1]
    ax.plot(xlist, ylist)
    ax.set_title('koch␣curve')
```

(To display within jupyter, execute `%matplotlib inline` at the beginning)

Problem **Riding the Bus⋆⋆** (World Finals 2003)

There are grid points on a Peano curve drawn within a square. Find the distance between two given points. The distance is the distance from the given point to the nearest grid point (or the one with the smallest x,y coordinates if there are multiple), and the distance along the Peano curve between the grid points. (The description of the error is a bit concerning)

https://icpcarchive.ecs.baylor.edu/index.php?option=com_onlinejudge&Itemid=8&category=37&page=show_problem&problem=724

Example solution: Divide the whole into 9 parts, and search in detail only in the relevant areas.

Problem	**Plotter**⋆⋆	(Algorithmic Engagements 2011)

When does the pen pass through a given point?
`http://main.edu.pl/en/archive/pa/2011/plo`

Chapter 7

Basic Data Structures (string, stack, queue, string, set, map)

Overview

In this chapter, we introduce strings, stacks, (priority) queues, sets, and associative arrays as ”tools” provided as standard libraries in many languages such as C++ and Java. Readers who find the introduction of tools boring can skip ahead and come back later when needed. In this material, we take the position of first recommending the use of standard libraries. Creating these tools yourself is useful as strength training, but it is not suitable for beginners. Also, in practical situations, it is preferable to use standard libraries if they are sufficient. For example, there are benefits such as expecting no bugs because they are used by many people, and they are suitable for collaborative work because other people know their properties well.
In this chapter, we will cover sample code for C++ and Python3. For Ruby, please refer to Appendix C.

7.1 Strings and Input/Output

C++

```
\begin{verbatim}
#include <iostream>
#include <string>
using namespace std;
int main() {
  string word; // Defines a string variable named word
  cin >> word; // Reads one word from standard input (delimited by spaces
or newlines)
```

```
  cout << word << word << endl; // Displays it twice
}
\end{verbatim}
```

If you input `hello`(newline), it will output `hellohello`.
The C++ string class is much more convenient than C-style strings, so it is recommended to get used to it early.

C++

```
\begin{verbatim}
#include <string>
  string word; // Definition
  string word2="ABCD"; // Definition and initialization
  word = "EF"; // Assignment
  string word3 = word + word2; // Concatenation
  char c = word[n]; // Extracts the n-th character
  word[n] = 'K'; // Assigns to the n-th character (fails if n<word.size() is not satisfied)
  cin >> word; // Reads one word (separated by newlines or spaces)
  cout << word.size() << endl; // Displays the number of characters
  if (word.find("A") != string::npos) ... // If word contains A...
\end{verbatim}
```

Python3

```
\begin{verbatim}
word = input().strip()
print(word*2)
\end{verbatim}
```

Reading one word and reading one line have different meanings, but we will not delve into the difference this time.

7.2 Stacks and Queues

Stacks and queues (Reference$^{\text{Strategy}}$[2, pp. 80–], Reference[3, pp. 31–]) are data structures for temporarily storing and retrieving data. Both manage data in a single line, storing new data at one end of the line (either end), and also retrieving data from one end (either end).

A stack is a data structure where the data stored later is retrieved first. For example, it is useful in situations where an urgent task arises while working, so the current task is placed at the top of a ”pile of work,” the urgent task is processed, and when that is finished, the process is resumed from the top of the pile of work. Of course, if another urgent task arises while processing an urgent task, it is similarly added to or removed from the pile of work.

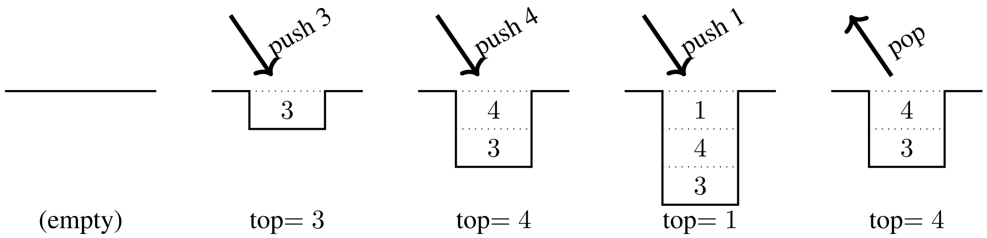
A queue is a data structure where the data stored earlier is retrieved first. This corresponds to the situation where people arriving at a restaurant line up at the back of the queue, and people are guided into the restaurant in the order they arrived earliest in the queue.

In a priority queue, data is retrieved in order of priority, not in the order they were queued. Although there is no perfect real-world example, it would correspond to a situation where patients are treated in order of priority from among those waiting, or a market where items can be obtained in order of the highest bid.

Note that in the C++ standard library, the retrieval operation is separated into two operations: referencing the first element `top()` or `front()` and removing the first element `pop()`[1]. In normal usage, we want to obtain the first element and remove it from the stack or queue, so we perform both operations consecutively.

7.2.1 Stacks

Overview A stack (stack) is a data structure that provides two operations: `push()` and `pop()`. These operations add and remove data, respectively. The data currently at the top can be referenced using `top()`.



Usage Notes Usually, arrays or `vectors` are used. The computational cost for typical implementations of `push()`, `pop()`, and `top()` is constant time. That is, it is expected that the computation time for each operation will not increase, no matter how much the data increases.

Note that while some implementations of `pop()` return a value like `top()`, in the C++ standard library, these are separated[2]. In normal usage, we want to get the first element and remove it from the stack or queue, so we perform both operations consecutively.

Also, member functions such as `size()` to check the current number of elements are provided, so please refer to a grammar book for details[3].

C++

```cpp
\begin{verbatim}
#include <stack>
#include <iostream>
using namespace std;
int main() {
    stack<int> S; // Stack to store ints
    S.push(3); // Add to the top
```

[1] This is for exception safety.

[2] This is for exception safety.

[3] There is also information on the web: http://en.cppreference.com/w/cpp/container/stack

```
    S.push(4);
    S.push(1);
    while (! S.empty()) { // While there are elements
        int n = S.top(); // Copy the top element
        S.pop(); // Discard the top element
        cout << n << endl; // Displays 1, 4, 3 in that order
    }
}
\end{verbatim}
```

In Python, we will use `append` and `pop` as `push` and `pop`, respectively.

Python3

```
\begin{verbatim}
stack = []
stack.append(3)
stack.append(4)
stack.append(1)
while len(stack) > 0:
    n = stack.pop()
    print(n)
\end{verbatim}
```

Example **Reverse Polish Notation** (AOJ)

Read a grammar given in ”Reverse Polish Notation” and calculate the value.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_3_A&lang=jp

Hint: Also refer to the ”Explanation” at the end of the problem statement. A detailed explanation is given in Reference$^{\text{Strategy}}$[2, p.82].

C++

```
\begin{verbatim}
#include <string>
#include <stack>
#include <iostream>
using namespace std;
int main() {
  string word;
  stack<int> S;
  while (cin >> word) { // Read as long as there is input
    if (word == "+") {
      // pop two numbers, push the sum
    }
    else if (word == "-") {
      // pop two numbers, push the difference
    }
    else if (word == "*") {
      // pop two numbers, push the product
    }
    else {
      // convert word to a number and push it
    }
  }
  // Display the top element of S.
}
\end{verbatim}
```

This program reads as long as the input continues. To give the end of input from the keyboard, use "^D" (press "d" while holding down the Ctrl key).

To convert a string representing a number to an integer `n`, in C++11, use `int n=stoi(word)`, otherwise, include `<cstdlib>` and use `int n=atoi(word.c_str())` etc. (AOJ's C++11 does not yet support `stoi`, so use `atoi`).

Problem **Largest Rectangle in a Histogram** (Ulm Local 2003)

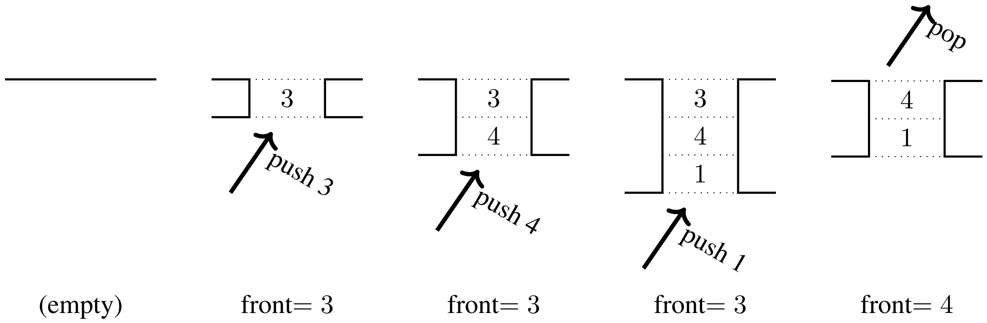
Find the area of the largest rectangle contained within a histogram. (Note: use long long)
http://poj.org/problem?id=2559

By skillfully utilizing a stack, it can be found in time proportional to the number of bars in the histogram.

Reference: http://algorithms.blog55.fc2.com/blog-entry-132.html

7.2.2 Queues

Overview A queue (queue,Reference[3, p. 32]) is a data structure that allows data to be stored (`push()`) and retrieved (`pop()`). The data currently at the front can be referenced using `front()`.



Usage Notes Arrays or deques are used for implementation. The computational cost for push(), pop(), and front() is constant time. That is, it is expected that the computation time for each operation will not increase, no matter how much the data increases. (Since moving the entire data would take $O(N)$, in practice, only indices or pointers representing the currently used section, such as head and tail, are manipulated).

In C++, a template class called queue is provided. The data that is put in (pushed) is retrieved by the front() and pop() operations.[4]

C++

```cpp
\begin{verbatim}
#include <queue>
#include <iostream>
using namespace std;
int main() {
    queue<int> Q; // Queue to store ints
    Q.push(3); // Add to the end
    Q.push(4);
    Q.push(1);
    while (! Q.empty()) { // While there are elements
        int n = Q.front(); // Copy the first element
        Q.pop(); // Discard the first element
        cout << n << endl; // Displays 3, 4, 1 in that order
    }
}
\end{verbatim}
```

In this material, we will use collections.deque's append, popleft as the representation of a queue in Python.[5]

Python3

[4] http://en.cppreference.com/w/cpp/container/queue

[5] https://docs.python.jp/3/library/collections.html#deque-objects

```
\begin{verbatim}
import collections
q = collections.deque()
q.append(3)
q.append(4)
q.append(1)
while len(q) > 0:
    n = q.popleft() # Retrieve from the front
    print(n)
\end{verbatim}
```

Example **Round-Robin Scheduling** (AOJ)

Let’s simulate the process of processing multiple calculations little by little.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_3_B&lang=jp

Hint: Also refer to the ”Explanation” at the end of the problem statement. A detailed explanation is given in Reference$^{\text{Strategy}}$[2, p.82].

Problem **Areas on the Cross-Section Diagram** (AOJ)

Output the area of the puddles that form when it rains on a given terrain.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_3_D&lang=jp

Hint: If a water surface is formed, it is the most recent pair of a downward slope and an upward slope at the same height. By looking at the string from left to right,

-
-

By doing this, it is possible to create pairs of downward and upward slopes at the same height. After that, it is good to remove the water surfaces that are hidden. For example, represent the start position and area with `pair<int,int>` and manage them with a `stack`.

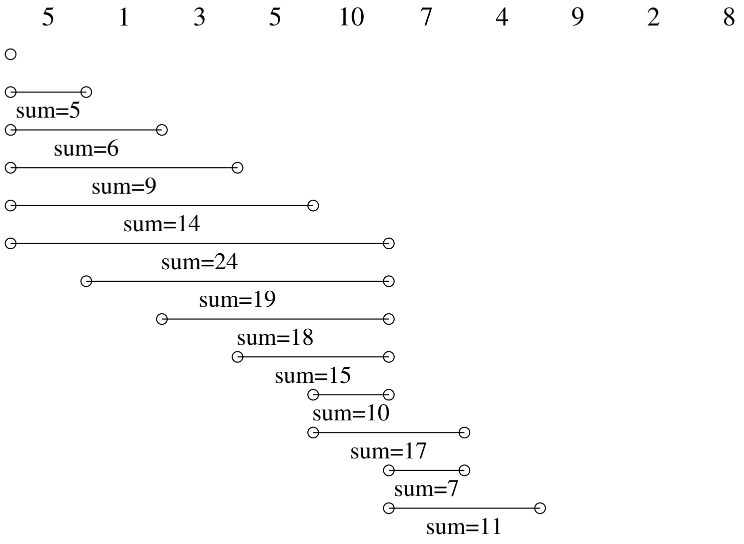
Problem **Subsequence** (Southeastern Europe 2006)

For a contiguous subsequence of array A, find the minimum length of a subsequence whose sum is greater than or equal to S.
`http://poj.org/problem?id=3061`

Explanation of the sample: The part (10,7) satisfies the condition $17 \geq 15$ and has the minimum number of elements, which is 2. The part (3,4,5) satisfies the condition $12 \geq 11$ and has the minimum number of elements, which is 3.

Hint: Checking all intervals takes $O(N^2)$, but by checking only the intervals where the sum is close to S as follows, it can be processed in $O(N)$. Look at the elements of the array in order from the first element, and if the sum of the elements inside the queue is less than S, `push` to increase the number of elements, and if it is greater than or equal to S, `pop` to decrease the number of elements. The sum of the elements inside the queue is managed by creating a variable to represent it and managing it at the timing of `push` and `pop`.

This is the so-called ”sliding window” technique. Since `cin` is probably slow, use `scanf`.



Problem **Sum of Consecutive Prime Numbers** (Asian Regional 2005)

Find the number of ways a given integer can be expressed as the sum of consecutive prime numbers.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1257&lang=jp

First, find the prime numbers up to 10 000 using the Sieve of Eratosthenes (Reference[3] page 112) or similar methods. After that, it is the same as ”Subsequence”.

7.2.3 Priority Queues

A priority queue (priority queue, Reference[3, p. 32]) is a data structure that allows data to be stored and retrieved, where the largest element among those not yet retrieved is retrieved first.

Example **Priority Queue** (AOJ)

Create a priority queue that can accept integers as input and retrieve them in descending order.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_9_C&lang=jp

First, it is good to confirm that you can get accepted using the standard library priority_queue[6]. If you implement a priority queue yourself, it is convenient to create a binary heap on an array or vector. See Reference$^{\text{Strategy}}$[2, Chapter 10, pp. 232–] for details.

C++

```cpp
\begin{verbatim}
#include <queue>
#include <iostream>
using namespace std;
int main() {
    priority_queue<int> Q; // Priority queue to store ints
    Q.push(3); // Add
    Q.push(4);
    Q.push(1);
    while (! Q.empty()) { // While there are elements
```

[6] http://en.cppreference.com/w/cpp/container/priority_queue

```
        int n = Q.top(); // Copy the first element
        Q.pop(); // Discard the first element
        cout << n << endl; // Displays 4, 3, 1 in that order
    }
}
\end{verbatim}
```

Since Python has a heapq package, we will use that.

Python3

```
\begin{verbatim}
import heapq

Q = []
heapq.heappush(Q, 3)
heapq.heappush(Q, 4)
heapq.heappush(Q, 1)
while len(Q) > 0:
    n = heapq.heappop(Q)
    print(n)                        # Displays 1, 3, 4 in that order
\end{verbatim}
```

7.3 Strings: Substrings, Concatenation, and Reversal

In C++, you can obtain substrings of string[7] data using the substr method[8].

C++

```
  string word = "hello";
  for (size_t i=0; i<=word.size(); ++i) { // size() is the length of the
string
    string l = word.substr(0,i);// String to the left of the i-th character
    string r = word.substr(i); // String from the i-th character onwards
    cout << l << '␣' << r << endl;
  }
```

Example output:

```
 hello
h ello
```

[7] http://en.cppreference.com/w/cpp/string/basic_string

[8] http://en.cppreference.com/w/cpp/string/basic_string/substr

```
he llo
hel lo
hell o
hello
```

Exercise: Try changing `string word = "hello";` to a different word and running it.

Python3

```python
word = "hello";
for i in range(len(word)+1):
    l = word[0:i]
    r = word[i:]
    print(l+'␣'+r)
```

For concatenation, use the + operator. It's the same in both C++ and Ruby.

C++

```cpp
    string a = "AAA";
    string b = "BBB";
    string c = a+b; // Concatenation
    cout << c << endl; // AAABBB
```

For reversal, use the `reverse` function. The range is specified by `word.begin()` and `word.end()`. Using a notation similar to arrays, you can also write `reverse(&word[0], &word[0]+word.size());`.

C++

```cpp
#include <algorithm>
    string word = "hello";
    reverse(word.begin(), word.end());
    cout << word << endl;
```

7.4 Sets

In C++, the standard library provides the data structures `set`[9] and `unordered_set`[10] (from C++11 onwards) to represent sets. First, let's introduce an example of a set of integers. Both provide almost the same operations, but here we will use insertion (`insert`), the total number of elements (`size`), and searching for the presence of a specified element (`count`).

In the implementation, `set` manages data with ordering, while `unordered_set` manages data ignoring the order. The former is usually implemented with a binary search tree, and insertion and search require $O(\log N)$ time. That is, note that one operation becomes slower as the number of data N increases. The latter is usually implemented using a hash table, and each operation is expected to take constant time on average.

[9] http://en.cppreference.com/w/cpp/container/set

[10] http://en.cppreference.com/w/cpp/container/unordered_set

C++

```
\begin{verbatim}
#include <set>
    typedef set<int> set_t;
    set_t A;
    cout << A.size() << endl; // 0 because it is empty at first
    cout << A.count(3) << endl; // 0 because 3 is not included
    A.insert(1);
    A.insert(3);
    A.insert(5);
    A.insert(3);
    cout << A.size() << endl; // 3 for 1, 3, and 5
    cout << A.count(3) << endl; // 1 (at most 1 in the case of set)
    cout << A.count(4) << endl; // 0
\end{verbatim}
```

A set of strings can be used similarly.

C++

```
\begin{verbatim}
#include <set>
#include <string>
    typedef set<string> set_t;
    set_t A;
    cout << A.size() << endl; // 0 because it is empty at first
    cout << A.count("hello") << endl; // 0 because "hello" is not included
    A.insert("hello");
    A.insert("world");
    A.insert("good␣morning");
    A.insert("world");
    cout << A.size() << endl; // "hello", "good morning", "world"
    cout << A.count("world") << endl; // 1 (at most 1 in the case of set)
    cout << A.count("hello!!!") << endl; // 0
\end{verbatim}
```

In Python, `set` is provided.[11]

Python3

```
\begin{verbatim}
s = set()
print('a' in s)                  # False
s.add('a')
```

[11] https://docs.python.jp/3/library/stdtypes.html#set

```
print('a' in s)                 # True
s.discard('a')
print('a' in s)                 # False
\end{verbatim}
```

Example **Search - Dictionary** (AOJ)

Determine whether a word is in the dictionary.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_4_C&lang=jp

Time Limit Exceeded?

If you do not get AC with a ”Time Limit Exceeded (TLE)” verdict, it means that the execution time exceeded the allowable range. If you are using `set` in C++, try using `unordered_set`. In that case, select C++11 instead of C++ in the submission window.

7.4.1 Bitsets

As a special case of sets, when dealing with sets of integers from 0 to N, using a bit representation is efficient. In C++, the `bitset` type is provided as standard.

C++

```
\begin{verbatim}
#include <bitset>
#include <cassert>
std::bitset<128> a, b, c;
a[3] = 1; // Set the specified bit to on
b[5] = 1;
c = a | b; // Union of sets
\end{verbatim}
```

7.5 Associative Arrays (map)

When managing names associated with personal identification numbers, using an array is common.

C++

```
\begin{verbatim}
string name[30];
name[0] = "kaneko";
name[1] = "fukuda";
\end{verbatim}
```

However, how can we manage personal identification numbers associated with names?

C++

```
\begin{verbatim}
number["kaneko"] = 0; // We want to write like this
\end{verbatim}
```

An associative array is an abstract data type that manages `values` associated with `keys`, providing the functionality described above. In C++, `map`[12] and `unordered_map`[13] are provided, and in Python, `dict`[14] is available as standard.

7.5.1 Maps

In C++, the standard library provides the data structures `map` and `unordered_map` (from C++11 onwards) to represent associative arrays. First, let's introduce an example of a map from strings to integers. Both provide almost the same operations, but here we will use insertion (using `[]`), the total number of elements (`size`), and searching for the presence of a specified key (`count`).

In the implementation, `map` manages data with ordering, while `unordered_map` manages data ignoring the order. The former is usually implemented with a binary search tree, and insertion and search require $O(\log N)$ time. That is, note that one operation becomes slower as the number of data N increases. The latter is usually implemented using a hash table, and each operation is expected to take constant time on average.

C++

```
\begin{verbatim}
#include <map>
#include <string>
#include <iostream>
using namespace std;
int main() {
    map<string, int> A;
    cout << A.size() << endl; // 0 because it is empty at first
    cout << A.count("hello") << endl; // 0 because "hello" is not included
```

[12] http://en.cppreference.com/w/cpp/container/map
[13] http://en.cppreference.com/w/cpp/container/unordered_map
[14] https://docs.python.jp/3/library/stdtypes.html#dict

```
    A["hello"] = 1;
    A["world"] = 3;
    A["good_morning"] = 5;
    A["world"] = 7;
    cout << A.size() << endl; // 3 for "hello", "good morning", and "world"
    cout << A.count("world") << endl; // 1 (at most 1 in the case of map)
    cout << A["world"] << endl; // 7
    cout << A.count("hello!!!") << endl; // 0
}
\end{verbatim}
```

In Python, dict is provided.[15]

Python3

```
\begin{verbatim}
d = {}
print('a' in d)                  # False
d['a'] = 1
print('a' in d)                  # True
print(d['a'])                   # 1
del d['a']
print('a' in d)                  # False
\end{verbatim}
```

Example **Search - Frequency** (AOJ)

Count the frequency of each word.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_4_B&lang=jp

Time Limit Exceeded?

If you do not get AC with a ”Time Limit Exceeded (TLE)” verdict, it means that the execution time exceeded the allowable range. If you are using map in C++, try using unordered_map. In that case, select C++11 instead of C++ in the submission window.

[15] https://docs.python.jp/3/library/stdtypes.html#dict

Problem **The Number of Palindromes** (Mock National Preliminary 2008)

Find the number of palindromic substrings in a given string.
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2018&lang=jp`

Hint:

- A palindrome is a string that reads the same forwards and backward.
- If the string is long, it is not possible to check all substrings.
- If the string is short, it is possible to check all substrings.
- If the string is a palindrome, the string with the first and last characters removed is also a palindrome.

Problem **The Number of Different Substrings** (Mock National Preliminary 2008)

Find the number of different substrings in a given string.
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2017&lang=jp`

Hint:

- If the string is long, it is not possible to check all substrings.
- If the string is short, it is possible to check all substrings.
- If the string is a substring, the string with the last character removed is also a substring.

Problem **The Number of Different Subsequences** (Mock National Preliminary 2008)

Find the number of different subsequences in a given string.
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2016&lang=jp`

Hint:

- If the string is long, it is not possible to check all subsequences.
- If the string is short, it is possible to check all subsequences.
- If the string is a subsequence, the string with the last character removed is also a subsequence.

Problem **The Number of Different Substrings** (Mock National Preliminary 2008)

Find the number of different substrings in a given string.
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2017&lang=jp`

Hint:

- If the string is long, it is not possible to check all substrings.
- If the string is short, it is possible to check all substrings.
- If the string is a substring, the string with the last character removed is also a substring.

Problem **The Number of Different Subsequences** (Mock National Preliminary 2008)

Find the number of different subsequences in a given string.
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2016&lang=jp`

Hint:

- If the string is long, it is not possible to check all subsequences.
- If the string is short, it is possible to check all subsequences.
- If the string is a subsequence, the string with the last character removed is also a subsequence.

7.6 Exercises

Problem **Queue** (AOJ)

Implement a queue using two stacks.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_3_C&lang=jp

Problem **Parenthesis Matching** (AOJ)

Determine whether parentheses are correctly matched.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_3_C&lang=jp

Problem **Area of a Polygon** (AOJ)

Find the area of a polygon.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=CGL_3_A&lang=jp

Problem **Convex Hull** (AOJ)

Find the convex hull of a set of points.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=CGL_4_A&lang=jp

Problem **Minimum Enclosing Circle** (AOJ)

Find the minimum enclosing circle of a set of points.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=CGL_7_A&lang=jp

Problem **The Number of Different Substrings** (Mock National Preliminary 2008)

Find the number of different substrings in a given string.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2017&lang=jp`

Problem **The Number of Different Subsequences** (Mock National Preliminary 2008)

Find the number of different subsequences in a given string.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2016&lang=jp`

Problem **The Number of Palindromes** (Mock National Preliminary 2008)

Find the number of palindromic substrings in a given string.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2018&lang=jp`

Problem **The Number of Palindromes** (Mock National Preliminary 2008)

Find the number of palindromic substrings in a given string.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2018&lang=jp`

Problem **The Number of Different Substrings** (Mock National Preliminary 2008)

Find the number of different substrings in a given string.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2017&lang=jp`

Problem **The Number of Different Subsequences** (Mock National Preliminary 2008)

Find the number of different subsequences in a given string.
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2016&lang=jp`

7.6.1 Mapping Strings to Numbers

C++

```cpp
\begin{verbatim}
#include <iostream>
#include <string>
#include <map>
using namespace std;
int main() {
  map<string,int> table; // Map to associate strings with numbers
  table["taro"] = 180;
  table["hanako"] = 160;
  cout << table["taro"] << endl; // 180
  cout << table["ichirou"] << endl; // 0
}
\end{verbatim}
```

In this example, the `key` is a string, and the `value` is an integer.

Python3

```python
\begin{verbatim}
all = {}
print(len(all))                    # 0
all["hello"] = 1
all["world"] = 100
all["good␣morning"] = 20
print(len(all))                    # 3
for k,v in sorted(all.items()):
    print(k,v)
# good morning 20
# hello 1
# world 100
\end{verbatim}
```

Example **English Sentence** (PC Koshien 2004)

Given a string, output the most frequent word and the word with the most characters.
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=0029&lang=jp`

The length of a string `s` of type `string` can be obtained with `s.size()`.

7.6.2 Counting String Occurrences

The behavior when a `key` does not exist depends on the processing system and settings, but in the above example, it is used so that it becomes 0. This makes it easy to count the number of occurrences.

C++

```
  table["ichirou"]+=1;
  cout << table["ichirou"] << endl; // 1
  table["ichirou"]+=1;
  cout << table["ichirou"] << endl; // 2
```

Python3

```
\begin{verbatim}
# Using a regular hash
table = {}
# table["ichirou"] += 1 ...  ng!
table["ichirou"] = table.get("ichirou", 0) + 1
if not "world" in table:
   table["world"] = 1
# Using Counter
import collections
c = collections.Counter()
c["ichirou"] += 1 # ok
\end{verbatim}
```

7.6.3 Listing Elements of Associative Arrays

C++11

```
\begin{verbatim}
#include <iostream>
#include <map>
#include <string>
using namespace std;
int main() {
  map<string,int> phone;
  phone["taro"] = 123;
  phone["jiro"] = 456;
  phone["saburo"] = 789;

  for (auto p=phone.begin(); // Equivalent to i=0
       p!=phone.end(); // Equivalent to i<N
       ++p) {
      cout << p->first << "␣" << p->second << endl;
  }
// The output order is
// jiro 456
// saburo 789
// taro 123
}
\end{verbatim}
```

The parts written in red are boilerplate, and since they are difficult to grasp at first glance, please use them as is. In the case of arrays, an integer `i` is used as an index, but in the case of associative arrays, because there is complex processing, an abstract interface called an iterator is used instead of an integer.[16] Also, for the initial and end values, `begin()` and `end()` are used instead of 0 and N. Inside the loop, the `key` of each element is referenced by `p->first`, and the `value` is referenced by `p->second`.

7.7 Exercises

Problem **Organize Your Train part II** (National Preliminary 2006)

How many ways are there to rearrange a freight train?
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1142&lang=jp

[16] Here, we used the C++11 feature and wrote `auto`. The actual type is `map<string,int>::iterator`.

C++

```cpp
int M;
string S;
int main() {
    cin >> M;
    for (int i=0; i<M; ++i) {
        cin >> S;
        std::set<std::string> all; // Set of strings
        all.insert(S); // Insert S into all
        for (size_t i=1; i<S.size(); ++i) {
            std::string L = S.substr(0,i); // Set L to the [0,i) characters of S
            std::string R = S.substr(i); // Set R to the [i,S.length()) characters of S
            std::string L2, R2;
            std::reverse(L.begin(), L.end()); // Set L2 to the reverse of L
            std::reverse(R.begin(), R.end()); // Set R2 to the reverse of R
            all.insert(R+L); // Insert R+L into all
            all.insert(R2+L); // Insert R2+L into all
            all.insert(L+R2); // Insert L+R2 into all
            all.insert(R2+L2); // Insert R2+L2 into all
            all.insert(L2+R); // Insert L2+R into all
            all.insert(L2+R2); // Insert L2+R2 into all
        }
        cout << all.size() << endl; // Output the number of elements in all (all.size())
    }
}
```

Problem **Era Name** (Mock Regional Preliminary 2010)

Given the correspondence between (fictitious) Western years and (fictitious) era names, organize and memorize them. Then, when a Western year is given as a question, if the era name is known, answer with the era name; otherwise, answer with “Unknown”.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2242&lang=jp

Problem Supplement: In the input example, ”showa 62 1987” represents two pieces of information:

- The Western year of the first year of Showa (year 1) is 1926.
- Showa continued (at least) until 1987.

Without other information, it is not known whether 1988 is Showa (another era name might be used between Showa and Heisei), so answer with “Unknown”.

Example Answer: Record each piece of information by dividing it into two: a correspondence table of Western years and the era name for which that year is year 1, and a table of how many years each era name lasted at most. Use `map<int,string>` for the former and `map<string,int>` for the latter.

7.7.1 Managing Intervals

Problem **Restrictive Filesystem**$\star$ (Mock National Preliminary 2009)

Implement read, write, and delete operations in a simple file system, and output the data read at each point in time.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2152&lang=jp

Example Solution:

7.7.2 Applications of Sets

Problem **Twenty Questions**$\star$ (Asian Regional 2009)

Consider the best way to ask questions to distinguish each object. The next question can be changed depending on the answer.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1302&lang=jp

Problem **Sweets**$\star$ (Algorithmic Engagements 2010)

There are $n \leq 24$ positive numbers. Divide them among three siblings A, D, B. Among the ways of dividing them that satisfy $A \geq D \geq B$, find the division that minimizes $A - B$.

https://szkopul.edu.pl/problemset/problem/cEMM4bdcKN06uikzs9BRSqkz/site/

Even the solution that the person in charge implemented, which is supposed to be the explained solution, gets TLE (time limit exceeded) (about 5 seconds for the maximum case), so it seems better to download the data from "Useful Resources" and not worry about the time if it is correct.

Problem **Building Blocks**⋆ (15th Polish Olympiad in Informatics)

There are N bar graphs. Select some consecutive $K(\leq N)$ bars and align them to the same height. Increasing or decreasing the height by 1 incurs the same cost. Output the minimum total cost and the height of each bar graph at that time.

`https://szkopul.edu.pl/problemset/problem/KC7c6nYfAXCbCGszqhIeOGxP/site/`

It is okay to use `cin` instead of `scanf` for input/output. It can be solved by combining STL data structures.

Chapter 8

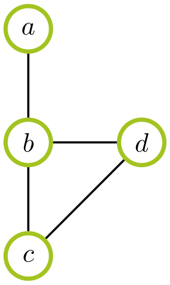
Graphs (1) Spanning Trees

Overview

Starting today, we will discuss graphs over several sessions. Graphs are a fundamental concept for modeling various situations and problems. This week, let's find the minimum spanning tree. Also, as a tool for that, we will introduce a data structure called Disjoint Set (Union-Find Tree) for managing groups.

8.1 Graphs and Trees

Graphs are often used when modeling the world with a focus on relationships. Examples include route maps, logistics, and family relationships. (Reference[3, Sections 2-5, p. 87] ”Everything is actually a graph”)



8.1.1 Graph Terminology

A graph consists of vertices (also called nodes or points, vertex; plural is vertices) and edges (also called branches, lines, edge, arc). A graph G is represented by a set of vertices V and a set of edges E as $G = (V, E)$.

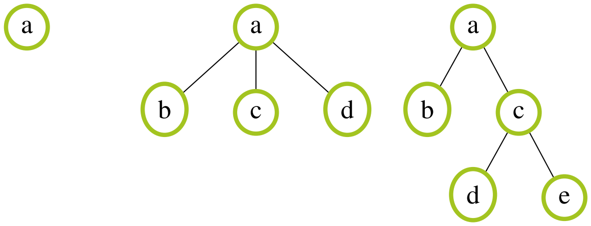
Example: A graph with three vertices $V = \{1, 2, 3\}$ where all vertices are connected (a triangle) has edges $E = \{\{1, 2\}, \{2, 3\}, \{3, 1\}\}$.

When a vertex v and an edge e satisfy $v \in e$, we say that v is **incident** to e. For a vertex v, the number of incident edges $|E(v)|$ is called the degree (degree) of v. Two vertices are said to be **adjacent** if they share a common incident edge. A sequence of adjacent vertices is called a path (path, trail, walk have slightly different meanings when studied in detail, so be careful when learning more).

Example: The degree of each vertex in a triangle is 2.

8.1.2 Trees

A special type of graph (connected and without cycles) is called a **tree**. Trees are easier to handle than general graphs. Examples of things that can be represented by trees include expressions (3 + (5 - 2)), hierarchical file systems (excluding links, etc.), family trees representing one's ancestors (when there are no marriages between relatives such as cousins), and Internet domain names.



Tree 1 Tree 2 Tree 3

Definitions For an undirected graph G, G is called **connected** if there exists a $v - w$ path for all pairs of vertices v, w. A graph that does not contain cycles is called a forest. A connected forest is called a tree.

If the number of vertices in a graph T is n, then T being a tree is equivalent to the following:

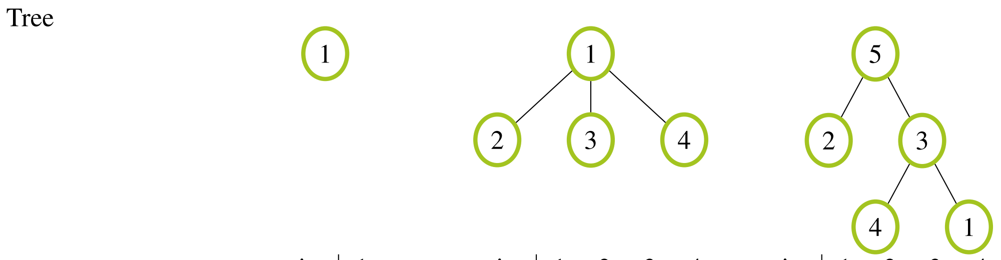
- T consists of $n - 1$ edges and has no cycles.
- T consists of $n - 1$ edges and is connected.
- For any two vertices in T, there exists only one path connecting the two vertices.
- T is connected, and for any edge in T, the graph obtained by removing that edge from T is disconnected.
- T contains no cycles, and for any two non-adjacent vertices x and y, the graph obtained by adding an edge connecting x and y to T contains a cycle.

A special vertex in a tree is called the root. When visualizing a graph, the root is often placed at the top or bottom. A vertex with a degree of 1 is called a leaf. For two vertices connected by an edge in a tree, the vertex closer to the root (or the root itself) is called the parent, and the vertex farther away is called the child.

8.1.3 Tree Representation Focusing on ”Parents”

One way to represent a ”tree” with N nodes on a computer is to assign numbers from 1 to N to each node and manage the parent of each node using a one-dimensional array (hereinafter denoted as `P[]`, using the initial of parent). Trees have the property that they have at most one parent. Therefore, for the value of `P[i]` corresponding to node i, we assign the parent's number if node i has a parent, and a special number such as -1 or itself if it does not have a parent (i.e., it is the **root**).

Note that in general graphs that are not trees, each node can have multiple children, so when managing children, a more complex representation than a one-dimensional array (such as an adjacency list or adjacency matrix, see Section 8.7) is required.



Array Representation

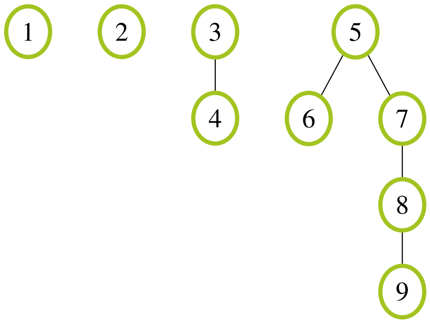
i	1
P[i]	1

i	1	2	3	4
P[i]	1	1	1	1

i	1	2	3	4	5
P[i]	3	5	5	3	5

8.2 Disjoint Set (Union-Find Tree)

One efficient method for determining whether two elements belong to the same group is the Union Find tree (Reference$^{\text{Strategy}}$[2, pp. 318–], Reference[3, pp. 81–]). It is also called a Disjoint-set data structure. It allows for the merging of groups but not their separation (c.f. link-cut tree).



i	1	2	3	4	5	6	7	8	9
P[i]	1	2	3	3	5	5	5	7	8

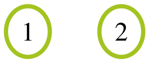
Four groups: $\{1\}, \{2\}, \{3, 4\}, \{5, 6, 7, 8, 9\}$

Determination If elements belong to the same tree, they are in the same group; otherwise, they are in different groups. This is determined by checking the root. Example:

- **Q1. Do 6 and 8 belong to the same group?**
  The root of the tree to which 6 belongs is 5, and similarly, the root of 8 is 5 ➔ Same group.

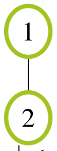
- **Q2. Do 2 and 4 belong to the same group?**
  The root of the tree to which 2 belongs is 2, and the root of 4 is 3 ➔ Different groups.

Merging Merging is the operation of combining two nodes into the same group. This is achieved by connecting the root of the tree to which one node belongs to the root of the tree to which the other node belongs. It does not matter which root is connected to which. (It is more efficient to attach the smaller (lower) tree under the larger (higher) tree, but this is omitted in this material.)



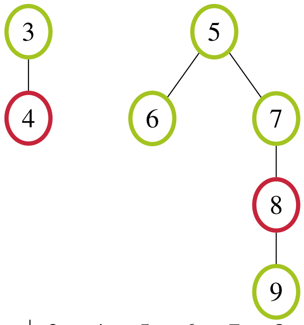
i	1	2
P[i]	1	2

Original graph



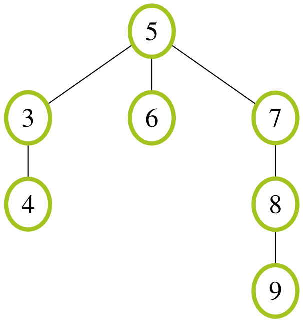
i	1	2
P[i]	1	*1*

Graph after merging 1 and 2



i	3	4	5	6	7	8	9
P[i]	3	3	5	5	5	7	8

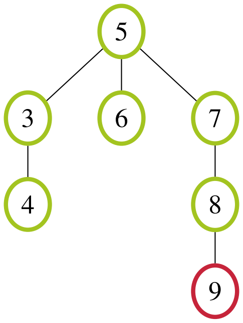
Original graph



i	3	4	5	6	7	8	9
P[i]	*5*	3	5	5	5	7	8

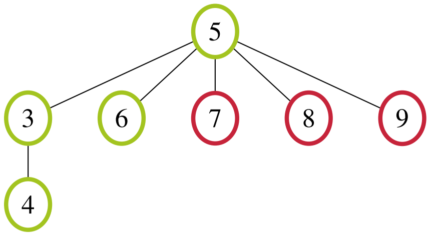
Graph after merging 4 and 8
(To merge the entire groups, connect the roots)

Optimization: Path Compression The operation of finding the root of a node takes longer as the distance from the root increases. Therefore, once the root is found, it is effective to improve efficiency by reconnecting the related nodes to directly connect to the root as much as possible.



i	3	4	5	6	7	8	9
P[i]	5	3	5	5	5	7	8

Original graph



i	3	4	5	6	7	8	9
P[i]	5	3	5	5	5	5	5

Deform the tree while finding root(9) = 5

Code Example: Below are code examples for initialization, determination, and merging. If it is difficult to understand the code at first glance, it is good to try writing it out and testing its operation before trying to understand it again. (The concept of `rank` is ignored.)

C++

```
\begin{verbatim}
int P[10010]; // Can handle vertices from 0 to 10000
void init(int N) { // Initialization:  Initially, all vertices are separate
    for (int i=0; i<N; ++i) P[i] = i;
}
int root(int a) { // Find the root (representative element) of a
    if (P[a] == a) return a; // a is the root
    return (P[a] = root(P[a])); // Find the root of a's parent and set it as a's parent
}
bool is_same_set(int a, int b) { // Do a and b belong to the same group?
    return root(a) == root(b);
}
void unite(int a, int b) { // Combine a and b into the same group
    P[root(a)] = root(b);
}
\end{verbatim}
```

The part where `P[a]` is assigned at the end of the `root` function is the path compression.

Python3

```
\begin{verbatim}
P = [i for i in range(N)]
```

```
def root(x):
    path_to_root = []
    while P[x] != x:
        path_to_root.append(x)
        x = P[x]
    for node in path_to_root:
        P[node] = x  # Path compression
    return x
def is_same_set(x,y):
    return root(x) == root(y)
def unite(x,y):
    P[root(x)] = root(y)
\end{verbatim}
```

In Python, the recursion depth limit is stricter than in C++, so the `root` function is implemented with a `while` loop instead of recursion.

Execution example:

C++

```
\begin{verbatim}
int main() {
  init(100);
  cout << is_same_set(1, 3) << endl;
  unite(1,2);
  cout << is_same_set(1, 3) << endl;
  unite(2,3);
  cout << is_same_set(1, 3) << endl;
}
\end{verbatim}
```

Example **Disjoint Set: Union Find Tree** (AOJ)

Manage groups with Disjoint Set.

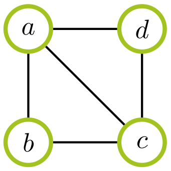
Note: Ignore the indices of the sets $S_1, \ldots, S_k$. In particular, it does not correspond to S_x in ”the set S_x containing x”.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=DSL_1_A&lang=jp

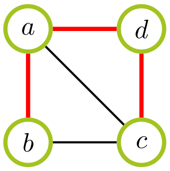
8.3 Spanning Trees

A tree formed using all the vertices of a connected graph G and all or a subset of its edges is called a spanning tree or a **full-vertex tree**. When the edges have weights, a tree where the sum of the weights of the edges included in the spanning tree is minimal is called a minimum (weight) spanning tree.

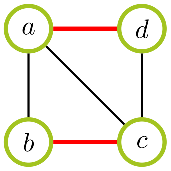
Note: If a graph is connected, a spanning tree exists. A spanning tree is not necessarily unique (in the case of a complete graph, there are n^{n-2} of them). A minimum spanning tree is also not necessarily unique. The problem of finding a minimum spanning tree often refers to the problem of finding one of the minimum spanning trees.



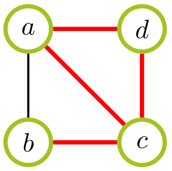
Original Graph



Red Subgraph is a Spanning Tree



Not a Spanning Tree (Disconnected)



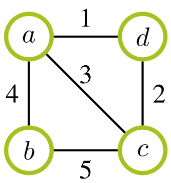
Not a Spanning Tree (Cy

8.3.1 Kruskal's Algorithm

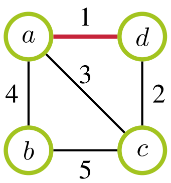
Among the well-known methods for finding the minimum spanning tree, there are Prim's algorithm and Kruskal's algorithm (Kruskal), but here we will introduce the latter.

Kruskal's algorithm operates as follows (Reference[3, pp. 101–]):

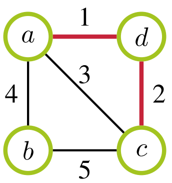
- Sort the edges in ascending order of weight (see Chapter 4.1.7 for how to sort pairs of numbers).
  - Method 1: Sort ⟨weight, start vertex, end vertex⟩
  - Method 2: Sort ⟨weight, edge ID⟩ and manage a separate array to associate IDs with start and end vertices.
- Let T be a forest in the process of being built (initially empty, a graph without cycles, not necessarily connected).
- For each edge e in ascending order of weight, perform the following operation: If $T + e$ (the graph obtained by adding edge e to graph T) does not contain a cycle, add e to T.



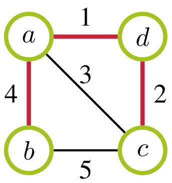
Original Graph



Adopt edge ad (weight 1)



Adopt edge cd (weight 2)



Adopt edge ab (weight 4)

One efficient method for determining whether $T + e$ contains a cycle is the union-find tree. Within the above Kruskal's algorithm, each time an edge e is added, the vertices at both ends of the edge are incorporated into the same group using `unite`. If `is_same_set(a,b)` is true for the two ends a, b of edge e, then adding edge e will create a cycle (because there is a path between a and b other than e).

Example **Graph II - Minimum Spanning Tree** (AOJ)

Find the sum of the weights of the minimum spanning tree of a given graph.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_12_A&lang=jp

Problem **Stellar Performance of the Debunkey Family** (PC Koshien 2008)

Problem: Minimize the maintenance cost of bridges while keeping the city connected.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=0180&lang=jp

Input/Output Example The following code can be created, for example, to read the input and output the edges in order of increasing maintenance cost. It is okay to copy this as is, since the main focus today is on practicing Kruskal's algorithm.

C++

```cpp
\begin{verbatim}
#include <algorithm>
int N, M, A[10010], B[10010], COST[10010];
pair<int,int> bridge[10010]; // Pair of cost and bridge number
int main() {
    while // (Read N and M, process if N is greater than 0)
        for (int i=0; i<M; ++i) {
            // Read A[i], B[i], and COST[i];
            bridge[i].first = COST[i];
            bridge[i].second = i;
        }
        sort(bridge, bridge+M); // Sort in ascending order of cost
        for (int i=0; i<M; ++i) {
            int cost = bridge[i].first;
            int a = A[bridge[i].second];
            int b = B[bridge[i].second];
            // Display "a bridge with cost connects a to b"
        }
    }
}
\end{verbatim}
```

In the code, `pair<int,int>` is equivalent to `struct pair { int first, second; };`.

Example Answer: Assuming the above preparation is done, the outline of the answer is as follows.

C++

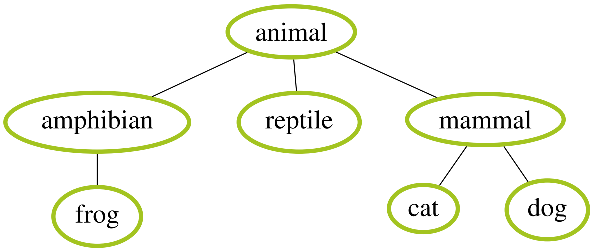
```
\begin{verbatim}
        int total = 0;
        for (int i=0; i<M; ++i) { // In order of cheapest bridge
            if // (If the nodes at both ends of the bridge are already connected)
                continue;
            // Merge the nodes at both ends of the bridge into the same group;
            // Add the cost of the bridge to the total;
        }
        // Output the total;
\end{verbatim}
```

8.4 Exercises

Problem **Marked Ancestor** (Summer Camp 2009)

Find the nearest marked ancestor. Initially, only the root is marked.
Note: The total may exceed the range of int, so use long long.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2170&lang=jp

Supplement: In the figure below, initially, the “nearest marked ancestor” of ”cat” is ”animal”. If ”mammal” is marked later, the “nearest marked ancestor” of ”cat” becomes ”mammal”.



Hint

First, read all the command sequences such as Mark and Query, and then,

8.4.1 Disjoint Sets

Problem **Fibonacci Sets** (Aizu University Programming Contest 2003)

Let $f[i]$ denote the i-th Fibonacci number. For some i, j ($1 \le i, j \le V$), if the absolute difference between $f[i]\%1001$ and $f[j]\%1001$ is less than d, then nodes i and j belong to the same group. Given V and d, count the number of groups.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1016&lang=jp

Example solution:

C++

```cpp
int F[1001];
int main() {
    F[0] = 1, F[1] = 2;
    // ...  Pre-calculate and assign the i-th Fibonacci number while (cin >> V >> D)
        // Initialize the tree
        for (int i=1; i<=V; ++i)
            for (int j=i+1; j<=V; ++j)
                if // (If the absolute difference between F[j] and F[i] is less than D)
                    // Unite group i and group j;
        // Count and output the number of i in [1,V] such that root(i) == i
        }
}
```

Create and test in detail:

- Calculation of Fibonacci numbers: Assign values to F[2]..F[1000] using a for loop (using `F` instead of `a[2]` in approach 3 of Section 12.4.1, and taking the remainder of 1001 during the calculation is the difference). That is, if you increase i from the smallest, you can calculate `F[i] = F[i-2]+F[i-1];` as defined. Take the remainder of 1001 even during the calculation to prevent overflow. (Display and confirm)
- Create a Union-find tree: Basically as in the material
  (Initialize, display the tree and confirm. Merge a few and display the tree to confirm.)
- Operation with Fibonacci numbers: Give a small V (for example, 10) and an appropriate D, display the tree, and test if it matches the manual calculation.
- Count the number of groups: Test the number of elements where root(i) == i.

Problem **True Liars** (Asian Regional 2002)

There is an island where people who only tell the truth (honest people) and people who only lie live, and the total number of each type is known. Given information that ”person x said that person y is/is not honest,” find the assignment if it is uniquely determined.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1238&lang=jp`

Hint

Even if it is not directly known whether a person is honest or not,

Problem **Never Wait for Weights⋆** (Asian Regional 2012)

(Paraphrased) Manage not only the groups to which elements belong but also the distances between elements.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1330&lang=jp`

Solution approach: It is sufficient to create a Union-find tree with the distance from the root added to the data.

Problem **Everlasting –One–⋆** (Asian Regional 2013)

Group jobs that can be changed to, and find how many groups there are that cannot be moved between.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2568&lang=jp

8.4.2 Various Spanning Trees

Problem **Slim Span⋆** (Asian Regional 2007)

Find a spanning tree where the difference between the minimum edge weight and the maximum edge weight is minimized.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1280&lang=jp

Hint

Consider the maximum edge weight of the spanning tree. Kruskal's algorithm outputs a tree where the maximum edge weight is minimized. Therefore, while trying all possible minimum weights a, examine the output of Kruskal's algorithm ignoring edges with weights less than a. Note that if the original graph is not connected, it may not be possible to create a spanning tree. It is good to confirm that the representative elements of the disjoint set are the same for all vertices when Kruskal's algorithm stops.

Problem **Sinking islands⋆** (Mock National Preliminary 2013)

Determine how to build bridges (see problem statement) on a sinking island.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2511&lang=jp

Problem **Byteland⋆** (1st Junior Polish Olympiad in Informatics)

If edges with the same weight exist, there can be multiple minimum spanning trees. For each edge, determine whether it can be included in a minimum spanning tree.

`https://szkopul.edu.pl/problemset/problem/JXaPu39wQfiZaqlM51zlchez/site/`

Although the time limit is set to a maximum of 15 seconds, it can be solved in about 1.5 seconds. Related to the proof that Kruskal's algorithm gives the correct solution.

Problem **There is No Alternative⋆** (Asian Regional 2014)

Find the edges that are always used in any minimum spanning tree.

`http://judge.u-aizu.ac.jp/onlinejudge/cdescription.jsp?cid=ICPCOOC2014&pid=F`

c.f. Similar problem `http://codeforces.com/problemset/problem/160/D`

8.5 Additional Notes: Other Topics Related to Trees

8.5.1 Tree Traversal

- **Preorder Traversal (Depth-First Search):** Visit the current node, then recursively visit the left subtree, and then the right subtree.
- **Inorder Traversal:** Recursively visit the left subtree, then visit the current node, and then the right subtree.
- **Postorder Traversal:** Recursively visit the left subtree, then the right subtree, and then the current node.

8.5.2 Tree Diameter

The diameter of a tree is the maximum distance between any two nodes in the tree. It can be found by performing a depth-first search (or breadth-first search) twice.

1. Perform a depth-first search from an arbitrary node s and find the node t that is farthest from s.
2. Perform a depth-first search from node t and find the node u that is farthest from t.
3. The distance between t and u is the diameter of the tree.

8.5.3 Lowest Common Ancestor (LCA)

The lowest common ancestor (LCA) of two nodes u and v in a tree is the deepest node that is an ancestor of both u and v. There are various methods for finding the LCA, such as using binary lifting or Euler tour techniques.

8.5.4 Tree Isomorphism

Two trees are said to be isomorphic if they have the same structure, even if the node labels are different. Determining whether two trees are isomorphic is a non-trivial problem. One approach is to use a canonical representation of the tree, such as a string representation based on tree traversal.

8.5.5 Minimum Spanning Tree (MST)

A minimum spanning tree (MST) of a weighted graph is a spanning tree with the minimum total edge weight.

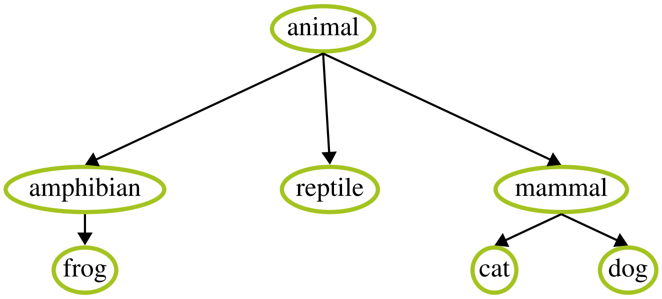
- **Kruskal's Algorithm:** Sort the edges by weight and add them to the MST if they do not create a cycle.
- **Prim's Algorithm:** Start with an arbitrary node and repeatedly add the minimum-weight edge that connects a node in the MST to a node outside the MST.

8.5.6 Applications of Trees

Trees are used in various applications, including:

- **Hierarchical Data Representation:** File systems, organizational structures, etc.
- **Search and Retrieval:** Binary search trees, tries, etc.
- **Decision Making:** Decision trees, game trees, etc.
- **Network Analysis:** Spanning trees, routing algorithms, etc.

8.5.7 Lowest Common Ancestor



For a given tree, the closest common ancestor of two nodes is called the lowest common ancestor (LCA). In the example figure, the LCA of dog and cat is mammal. The LCA of frog and dog is animal.

Example **Nearest Common Ancestors** (Taejon 2002)

Given a tree and two nodes in the tree, find the closest common ancestor of the two nodes.
`http://poj.org/problem?id=1330`

Let's try solving it by dividing it into several steps:

1. Understand the input visually. There is a number T on the first line, followed by T sets of ”tree information and two nodes for which to find the LCA”. Write down the tree in the Sample Input on paper.

2. Read the tree input. Display each `P[]` and confirm. Does the parent of each node match the tree drawn on paper?

3. As preparation for finding the LCA, consider finding all the nodes from a certain node x to the root. The parent of x can be found with `P[x]`, so by following the parent of the parent, the parent of the parent of the parent, and so on, we will eventually reach the root. Display the nodes when following the parent and confirm.

4. For each of nodes A and B, find the common elements of the paths (node number sequences) to the root. The one farthest from the root is the answer we are looking for.

As an advanced topic, if you need to find the LCA multiple times for one tree, it can be found efficiently in $O(\log N)$ per query by preparing in advance (see Chapter 16.3 or Reference[3, p. 274]).

Problem **Apple Tree**⋆ (POJ Monthly–2007.08.05)

Count the number of apples.
`http://poj.org/problem?id=3321`

8.5.8 Moving to the Parent

Problem **Marbles on a tree** (Waterloo local 2004.06.12)

Given a tree with N nodes. Each node in the tree has either no fruit or one or more fruits. The total number of fruits is N. What is the minimum number of moves required to ensure that each node has exactly one fruit? Moving one fruit along one edge counts as one move.
`http://poj.org/problem?id=1909`

Here is a sample code for input:

C++

```
\begin{verbatim}
#include <cstdio>
using namespace std;
int N, P[10010], M[10010], V[10010];
int main() {
    while (~scanf("%d", &N)) {
        fill(P, P+N, -1);
        int /*number of leaves*/L=0, id, c;
        for (int i=0; i<N; ++i) {
            scanf("%d", &id);
            scanf("%d", &V[id]);
            for (int j=0; j<V[id]; ++j) {
                scanf("%d", &c);
                P[c] = id;
            }
            if (V[id] == 0) ++L; // If V is 0, then id is a leaf
        }
        for (int i=0; i<N; ++i) {
            // Try outputting the parent
            printf("P[%d]=%d\n", i, P[i]);
        }
    }
}
\end{verbatim}
```

(Since some problems on poj have time limits when using cin, scanf is recommended)

Strategy: For each node, consider the fruits moving from/to the parent. For example, it is wasteful to receive 3 from the parent and return 5, so it is better to just send 2 after the difference. In other words, for each node, it is good to first adjust the shortage and surplus among the children, and then ask the parent to adjust the remaining amount.

C++

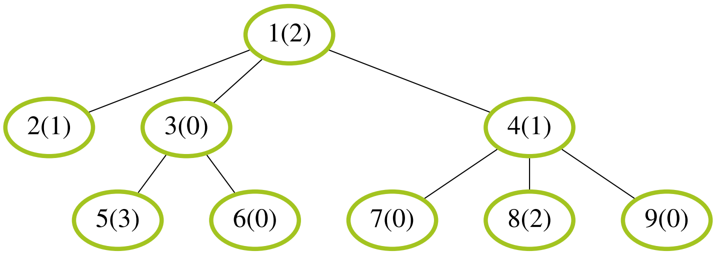


Figure 8.1: Sample 1

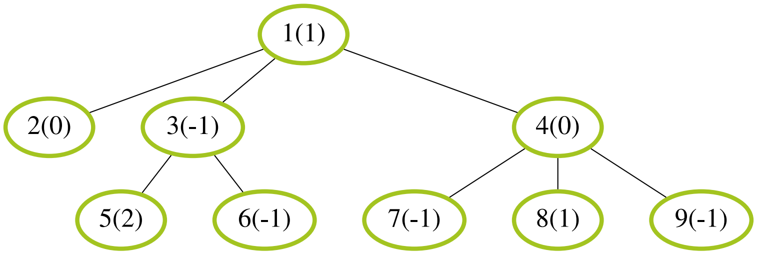


Figure 8.2: Sample 1 (Surplus/Shortage)

```
\begin{verbatim}
  // Prepare an array to indicate whether a node has been adjusted.  Initially, set the leaves as adjusted.
  // Prepare an array to represent the surplus or shortage at each node. The initial value is the number of fruits placed - 1.
  while // The root is not adjusted
    for // all nodes i
      if // i is not adjusted and all children of i are adjusted
        // Push the surplus or shortage at i to the parent
        // Record i as adjusted
      }
    }
  }
  // The sum of the absolute values of the surplus or shortage at each node is the answer
\end{verbatim}
```

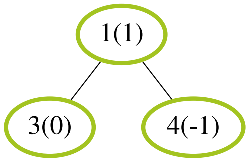


Figure 8.3: Adjusted Leaves (2=0, 5=2, 6=-1, 7=-1, 8=1, 9=-1 are removed)

8.5.9 Trees and Dynamic Programming

It is appropriate to tackle this section after becoming familiar with graph searching in Chapter **??**.

Problem **TELE**⋆ (Croatia OI 2002 Final Exam - Second Day)

(Paraphrased) A broadcasting station is planning to distribute programs. The amount viewers pay when they can receive the program is given in advance. The distribution network has a tree structure. The nodes of the tree are the broadcasting station itself at the root, viewers at the leaves, and relay devices elsewhere. How many people can receive the broadcast without incurring a loss?

(Author's note) There are no constraints on each cost C, but it seems to be positive and less than or equal to 5000.

`http://poj.org/problem?id=1155`

Hint: Perform a depth-first search from the root while managing something like `T[i][j]`, the cost of distributing to `j` people from node `i`. At each node, for example, at an internal node with two children, the revenue when distributing to 3 people needs to be maximized from all allocations such as (0,3), (1,2), ... (3,0).

Problem **Heap**⋆⋆ (POJ Monthly–2007.04.01)

There is a binary tree with integers written on the nodes. We want to rewrite the integers so that for any node, (left descendants) $<$ (right descendants) $\leq$ (itself) is satisfied. What is the minimum number of places that need to be rewritten?

The tree is not necessarily a complete binary tree, and the latter part may not be filled.

`http://poj.org/problem?id=3214`

Note the input format.

8.5.10 Tree Diameter

The diameter of a tree is the maximum distance between any two nodes in the tree. It can be found by performing a depth-first search (or breadth-first search) twice.

1. Perform a depth-first search from an arbitrary node s and find the node t that is farthest from s.
2. Perform a depth-first search from node t and find the node u that is farthest from t.
3. The distance between t and u is the diameter of the tree.

Problem **Fuel**⋆ (Algorithmic Engagements 2011)

Given a tree and the number of edges that can be traversed, design a sightseeing tour that maximizes the number of vertices visited (walk; the same edge can be traversed multiple times, the start and end points can be different).

https://szkopul.edu.pl/problemset/problem/5g0vDW-MvMGHfWQqh56jQKx1/site/

Problem **Bridge Removal**⋆ (National Preliminary 2014)

Find the shortest time for a craftsman to remove all bridges while traversing them in an archipelago.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1196&lang=jp

8.5.11 Tree Normalization

Problem **Professor Bubei, or How We Became Asymmetrical**⋆⋆ (National Preliminary 2007)

Given a binary tree representing an expression, normalize it according to the given rules.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1152&lang=jp

(This is a challenging problem, so it is for experienced users. Also, in ICPC, it is required to solve problems as quickly as possible to minimize terminal occupation time.)

8.5.12 Tree Center

Problem **Cave⋆⋆** (11th Polish Olympiad in Informatics)

A treasure is hidden at some node in a tree. Find the treasure based on the answers to questions. Determine the maximum number of questions needed with a question strategy that minimizes the number of questions.

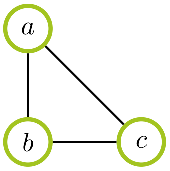
A question involves selecting one node and asking if the treasure is there. If the treasure is there, the game ends. Otherwise, the direction in which the treasure is located is indicated by an adjacent node.

`https://szkopul.edu.pl/problemset/problem/5Z9PRRPP-R90WhmbSY_qHd-1/site/`

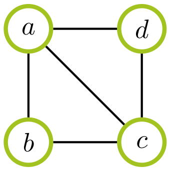
8.6 Other Graph Concepts

A closed path/trail that traverses each edge of a graph exactly once is called an Euler circuit/trail (Eulerian circuit/Eulerian trail).

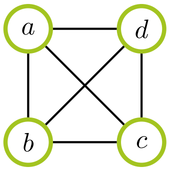
Criteria: For a circuit, the graph must be connected, and all vertices must have an even degree. For a trail, all vertices except the start and end points must have an even degree.



Has an Euler circuit (e.g., abca)



Can traverse all edges but no circuit (adcabc)



Cannot traverse all edges

Example **Patrol** (PC Koshien 2005)

Problem: Determine whether the streets to be patrolled can be traversed in a single stroke. The start and end points are specified. It is guaranteed that the graph is connected.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=0086&lang=jp`

Example Solution

C++

```
\begin{verbatim}
int main() {
    while (cin >> a >> b) { // Read edges a,b
        ... // Increment the degree of vertex a
        ... // Increment the degree of vertex b
        if (a == 0) {
            ... // Determine and output whether a single stroke is possible from the start to the end point (*)
                // (*) The degrees of vertices 1 and 2 are odd, and
                // the degrees of all vertices with numbers 3 or greater are even
            // Reset the degrees of all vertices to 0 for the next test case
        }
    }
}
\end{verbatim}
```

Problem **Play on Words** (Central Europe 1999)

Given a set of words, determine whether all the words can be connected in a ”Shiritori” (word chain) sequence. The first and last words can be chosen freely. (It is not guaranteed that the graph is connected)

Author’s note: Since the time limit is strict, it is better to use scanf for input.

http://poj.org/problem?id=1386

Example Solution Consider a graph where the alphabets are vertices. For example, consider the word news as an edge from vertex n to s. Here, since it cannot be used in the opposite direction, the edges (unlike before) have a direction (called a directed graph).

The degree of a directed graph is handled by distinguishing between **in-degree** and **out-degree** according to the direction of the edge.

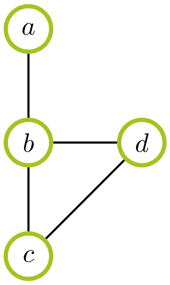
The necessary and sufficient conditions for a directed graph to have an Eulerian circuit are that it is connected and that the in-degree and out-degree of all vertices are equal. In the case of a non-circuit, the in-degree is one more/less than the out-degree at the start/end point of the path.

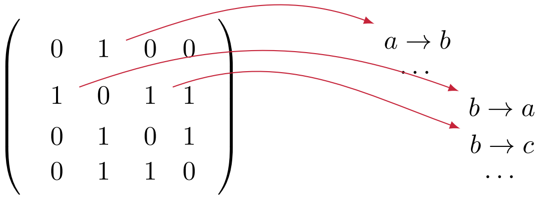
Unlike Problem A, connectivity is not guaranteed, so it must be determined manually (see Chapter **??**).

8.7 Graph Representations: Adjacency Lists and Adjacency Matrices

While remembering the parent of each node allowed us to traverse the tree structure by following the parent, more convenient data is needed to visit nodes in a general graph.

Adjacency Matrix First, we introduce a method to represent a graph using an adjacency matrix. When an edge exists from node i to j, the value at row i, column j of the matrix is set to 1; otherwise, it is set to 0. When dealing with undirected graphs, where edges have no direction, the matrix becomes symmetric.





$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

If we associate the nodes a, b, c, d of the left graph with the numerical values $0, 1, 2, 3$, respectively, the adjacency matrix becomes as shown on the right. That is, rows $0, 1, 2, 3$ represent the edges from a, b, c, d, respectively, and columns $0, 1, 2, 3$ represent the edges to a, b, c, d, respectively.

Note that we may use values other than 1 for the matrix elements in the future to represent the cost of edges or the number of paths.

Among graph representations, the adjacency matrix is a relatively ”luxurious” method. If the number of cities is N, it always uses memory proportional to N^2.

Adjacency List An adjacency list represents the list of destination vertices as a list, as shown in the figure. In practice, it is easier to manage the vertex names a, b, c, d using vertex numbers $0, 1, 2, 3$. In C++, if the number of vertices is N, we can use `vector<int> edges[N];` so that `edges[i]` represents the destinations of the i-th vertex. When managing the cost of edges, as in the previous chapter, we can use pairs of destinations and costs, such as `vector<pair<int,int>> edges[N];`.

$$\begin{array}{cclll} a & \rightarrow & b & & \\ b & \rightarrow & a & c & d \\ c & \rightarrow & b & d & \\ d & \rightarrow & b & c & \end{array}$$

An adjacency list uses memory proportional to the number of edges. If the graph is sparse, that is, if the number of edges is significantly less than N^2, the adjacency matrix has many elements with a value of 0, which is wasteful.

For example, in the case of a ”tree,” the number of edges is only $N-1$. Also, in real-world graphs, such as train route maps, the number of edges is often sparse compared to the number of vertices. Adjacency lists are suitable for such graphs. (Reference: Reference$^{\text{Strategy}}$[2, pp. 264–(Chapter 12)], Reference[3, pp. 90, 91])

Example **Graph** (AOJ)

Given an adjacency list of a directed graph, convert it to an adjacency matrix.

First, the number of nodes N in the graph is given, followed by N lines. Each line corresponds to one node and represents the edges coming out of that node. The first number u is the node number, the next number k is the number of edges, and the following k numbers correspond to the destinations of each edge. (The number of numbers included in each line may vary.)

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_11_A&lang=jp

In many cases, as in this problem, the vertex numbers are given as $1, \ldots, N$. On the other hand, in C++ and other languages, array indices start from 0, so in this material, we subtract 1 from the vertex numbers and handle them internally as $0, \ldots, N-1$, and add/subtract 1 when inputting/outputting. An example answer in Python is shown below. In this material, variables that appear in the problem statement are written in uppercase as constants (when there is no need to change their values later).

Python3

```python
N = int(input())
G = [[0 for _ in range(N)] for _ in range(N)] # NxN 2D array
for _ in range(N):
    u,k,*varray = map(int,input().split()) # u,k are numbers, varray is an array
    for v in varray:
        # Connect u-1 and v-1 in G
for row in G:
    print(’␣’.join(map(str,row))) # Output each row of G
```

8.8 Breadth-First Search (BFS)

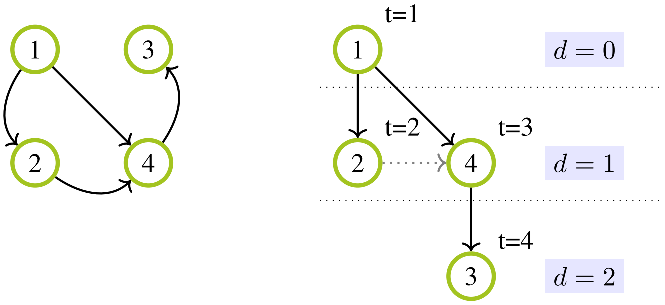
Problem **Breadth First Search** (AOJ)

Perform a breadth-first search starting from node 1 and display the distance of each node from the starting node.

(The input format is the same as the example problem “Graph”)

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_11_C&lang=jp`

Breadth-first search (Reference$^{\text{Strategy}}$[2, pp. 282–], Reference[3, p. 36]) visits vertices in order of their distance from the starting point.



Input Graph (Sample Input) Breadth-First Search Tree

Assuming the graph shown on the left in the figure, which is the sample input for the problem, is given as input, the right side shows an example of performing a breadth-first search starting from vertex 1. The subscript t of each node in the tree on the right represents the visit time, the solid lines represent the edges used during the breadth-first search, and the dotted lines represent the ignored edges. d is the distance from the starting point. In breadth-first search, the order of visiting vertices with the same distance is arbitrary.

To achieve such a search, we use a data structure called a queue (see Chapter 7.2.2). Data that is put (pushed) into the queue is retrieved in the order it was put in, using the `front()` and `pop()` operations.

1. Set the distance d_n from the starting point to each vertex n to ∞.
2. Put the starting point into the queue. Set the distance from the starting point (to itself): $d_{\text{start}} = 0$.
3. While the queue is not empty, repeat the following:
   (a) Remove the vertex n at the front of the queue.
   (b) For each vertex n' that can be reached from n, if $d_{n'} = \infty$ (meaning it is a newly discovered vertex), do the following:

i. Put n' into the queue.

ii. Set the distance to n': $d_{n'} = d_n + 1$.

The input format for this problem is the same as the example problem “Graph”, and the sample input is also exactly the same, so the input is already created, and we assume that when `G[s][t]==1` (and only then), it is possible to move from node `s` to node `t`.

Python3

```
 1  import collections
 2  D = [-1 for _ in range(N)]
 3  D[0] = 0 # The distance to the starting point is 0, other distances are
    -1
 4  Q = collections.deque()
 5  Q.append(0)                         # Starting point
 6  while len(Q) > 0:
 7      print("bfs", Q) # Check the operation of Q at each step (remove later)
 8      cur = Q.popleft()
 9      for dst in range(N):
10          if ...: # If it is possible to move from cur to dst and dst has
    not been visited
11              D[dst] = D[cur]+1
12              Q.append(dst) # Push dst onto Q
13  for v in range(N):
14      print(v+1, D[v])         # Convert from [0,N-1] to [1,N]
```

The execution example is as follows:

```
bfs deque([0])
bfs deque([1, 3])
bfs deque([3])
bfs deque([2])
1 0
2 1
3 2
4 1
```

During the BFS process, the vertices that can be reached from the starting point are put into the queue in order, each only once. In the condition on line 10, if we do not check whether `dst` has been visited, problems will occur in graphs with merges or loops (for example, if there is an edge from 1 to 2 and also an edge from 2 to 1, it will continue to move 1-2-1-2-1 and loop infinitely). Whether or not it has been visited can be determined by looking at `D[dst]`.

C++

```
#include <queue>
int D[...];
void bfs(int src) {
    cerr << "bfs␣root␣=␣" << src << endl;
    queue<int> Q; // Definition of a queue to manage integers
    Q.push(src);
    D[src] = 0; // Starting point
    while (! Q.empty()) {
        int cur = Q.front(); // Retrieve the first element
        Q.pop();
        // Display for operation check
        cerr << "visiting␣" << cur << '␣' << D[cur] << endl;
        for (...) { // For each destination dst
            if (..) { // If there is an edge from cur to dst and dst has not been visited
                D[dst] = D[cur]+1; //
                Q.push(dst); // Add dst to the visit destinations
            }
        }
    }
}
```

8.9 Depth-First Search (DFS)

Example **Depth First Search** (AOJ)

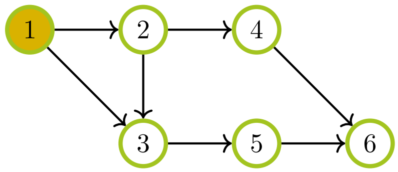
Display the visit order of nodes when performing a depth-first search, visiting nodes in ascending order of their numbers.

Note: ”If there are any unvisited vertices remaining, continue the search by using one of them as a new starting point.”

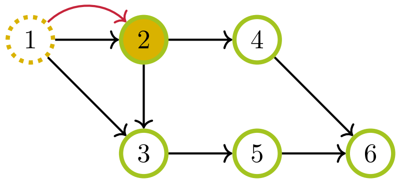
(The input format is the same as the example problem “Graph”)

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_11_B&lang=jp

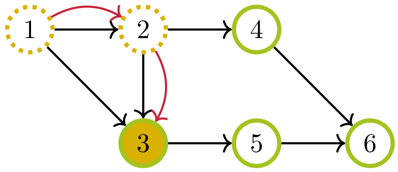
Depth-first search (Reference$^{\text{Strategy}}$[2, pp. 273–], Reference[3, p. 33]) is another method for visiting all nodes, visiting nodes in order from those closest to the currently visited node. First, we will introduce an implementation using a recursive procedure.



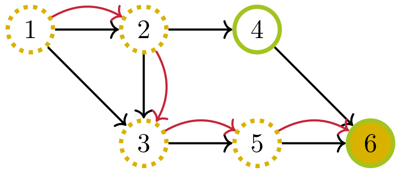
Time 1: Start from vertex 1



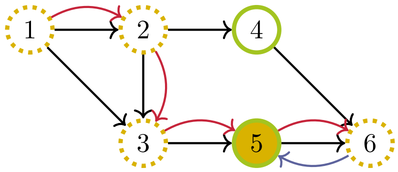
Time 2: Select 2 from destination candidates 2 and 3



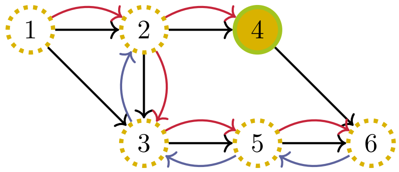
Time 3: Similarly, select 3



Time 5: Similarly, proceed to 6, and there are no more destinations



Time 7: Return to parent



Time 9: If there is an unvisited destination, proceed

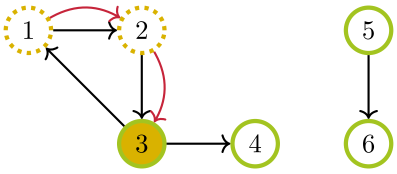
C++

```
\begin{verbatim}
int time = 0;
void dfs(int cur) { // Visit cur
    // Record the visit time of cur
    time += 1;
    // Display for operation check
    cerr << "visiting␣" << cur << '␣' << time << endl;

    for (int dst = ...; ...; ...) { // For all nodes dst
        if (...) { // If there is an edge from cur to dst and dst has not been visited
            dfs(dst);
        }
    }
    // Record the departure time of cur
    time += 1;
    // Return to the parent (blue arrow) at the end of the function
}
\end{verbatim}
```

Python3

```
\begin{verbatim}
time = 1
D = [-1 for _ in range(N)]
F = [-1 for _ in range(N)]
def dfs(src):
    global time # Declare that the global variable will be rewritten within the function
    D[src] = time # Record the visit time
    time += 1
    for dst in range(N):
        if ...: # If it is possible to move from src to dst and dst has not been visited
            dfs(dst)
    F[src] = time # Record the departure time
    time += 1
    # Return to the parent (blue arrow) at the end of the function

for root in range(N): # From the node with the smallest number
    if ...: # If D[root] has not been visited
        dfs(root) # Start dfs
# Output
\end{verbatim}
```

Detection of Loops and Disconnected Graphs



It is necessary to assume in advance what kind of graph is targeted, and in this problem, graphs like the one above can also be given. First, be careful not to proceed to vertex 1 when you have proceeded to vertex 3 (it will become an infinite loop). To prevent this, it is good to check the visit time of the destination candidates and identify whether they have already been visited. Such edges that return to visited vertices are sometimes called back edges.

Also, in this problem, after finishing the DFS starting from vertex 1, it is required to start the DFS from vertex 5 next, and continue until all vertices have been visited. After finishing the DFS, check if there are any unvisited vertices while increasing the vertex number, and if found, perform DFS starting from there.

DFS Using a Stack Explicitly (Reference)

Usually, the recursive implementation introduced earlier is sufficient. However, if the number of recursion levels becomes deep, the stack area (distinguish from the stack of data structures) allocated to the process may become insufficient, causing a segmentation fault or similar. In practice, it is often sufficient to increase the allocation of the stack area using a method appropriate for the environment, but in contests, it is often not possible to use such means.

The following shows an implementation using a stack of data structures (Chapter 7.2.1). The data put (pushed) into the stack is retrieved in the reverse order it was put in, using the `top()` and `pop()` operations. The main part is just replacing the queue of BFS with a stack (however, the actual search behavior differs greatly due to the difference between queue and stack).

C++

```
\begin{verbatim}
void dfs(int src) {
    // cerr << "dfs root = " << src << endl;
    stack<int> S;
    S.push(src);
    while (! S.empty()) {
        int cur = S.top();
        S.pop();
        if (...) { // If cur is a first visit
            // Record the first visit time
            S.push(cur); // (*) After visiting all descendants, return to self once more
            for (...) { // For all child nodes dst
                // If there is an order among siblings, set the for loop in the reverse order of the desired visit order
                if (...) { // If there is an edge from cur to dst and dst has not been visited
                    S.push(dst); // Add dst to the todo list
                }
        }
        else if (...) { // This is the second visit to cur
          // Since the descendants corresponding to (*) have been visited, record the departure time of cur
        }
        else if (...) { // This is the third or later visit to cur
          // Do nothing (a case where it was scheduled to be visited later,
          // but was visited earlier via a descendant node)
        }
    }
}
\end{verbatim}
```

If it is sufficient to visit each node once, the `push` on line 11 is unnecessary. This time, since we need not only the visit time but also the departure time, we are putting each node into the stack twice, once for the visit and once for the departure. Also, the `for` statement on line 12 needs to have its order adjusted, considering that the problem requires visiting nodes in ascending order of their numbers and that the stack retrieves data in the reverse order it was put in.

8.10 Connectivity Determination

Whether a graph is connected can be determined using either BFS or DFS.

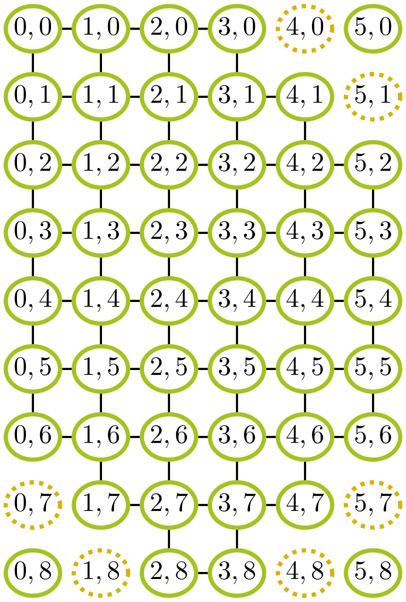
In the following problems, the edges and vertices of the graph are not explicitly given in the problem statement. However, a solution can be obtained by constructing the graph yourself and performing a search.

Problem **Red and Black** (National Preliminary 2004)

Find the number of reachable squares by moving only up, down, left, and right.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1130&lang=jp`

Consider each square as a vertex, and create edges between adjacent vertices that can be moved to. There is no need to assign serial numbers to the vertices.



Up, Down, Left, and Right of Square (x,y) The squares adjacent up, down, left, and right are (x+1,y), (x-1,y), (x,y+1), and (x,y-1). However, it is necessary to be careful not to go out of bounds of the map.

Representation of Directions In practice, it is better to avoid writing the up, down, left, and right movements by hand, as it is a source of bugs. By preparing an array like `const int dx[]={1,0,-1,0}, dy[]={0,-1,0,1};`, the parts where similar 4 lines are arranged in the search can be grouped together using a for loop. That is, one of the squares adjacent to (x,y) is (x+dx[i], y+dy[i]).

Whether to Move to a Square It is convenient to create a function to check whether it is possible to move to (x,y), which checks (1) whether it is within the bounds of the map and (2) whether it is not a wall. Test in the order of (1) and (2).

C++

```
\begin{verbatim}
bool valid(int x, int y) {
   return x is in the range [0,W]
        && y is in the range [0,H]
        && (x,y) is not a wall;
}
\end{verbatim}
```

Outline of the Answer: Find the position (x,y) of '@', and from there, the number of vertices that can be visited by depth-first search or breadth-first search is the answer.

8.11 Bipartite Graph Determination

A graph where the vertices can be colored with two colors (for example, red and blue) such that the colors at both ends of each edge are different is called a bipartite graph. To determine whether a given graph is bipartite, traverse the edges while coloring with BFS or DFS and check for contradictions.

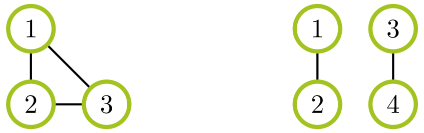
Problem **A Bug's Life** (TUD Programming Contest 2005)

There are bugs whose genders are unknown. Given information that ”bug i and bug j are of opposite genders,” determine whether there are any contradictions.

(Or) Determine whether the graph is bipartite.

http://poj.org/problem?id=2492

Interpretation of the sample input:



Inconsistent Consistent

Data storage:

C++

```
% Maximum number given in the problem statement
int bugs, edges;
% Adjacency matrix: if e[i][j] is true, there is an edge between i<->j
bool e[2010][2010];
% Color of each bug 0: undecided, 1,-1: male/female
int color[2010];
```

Input/output example:

C++

```
% Assigns color \texttt{id\_color} to bug id and returns whether it is consistent
% Consistent \dingright{} true, inconsistent \dingright{} false
bool search(int id, int id_color) {
  % Create three versions here
}
```

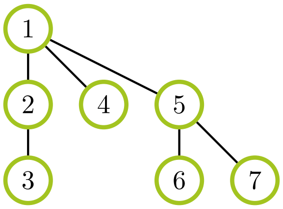
C++

```
int main() {
    int scenarios;
    scanf("%d", &scenarios);
    for (int t=0; t<scenarios; ++t) {
        % This for loop block is one problem
        fill(&e[0][0], &e[0][0]+2010*2010, 0); % Initialize to 0
        fill(&color[0], &color[0]+2010, 0); % Initialize to 0
        scanf("%d␣%d", &bugs, &edges);
        for (int j=0; j<edges; ++j) {
            int src, dst;
            scanf("%d␣%d", &src, &dst);
            e[src][dst] = e[dst][src] = 1; % Make bidirectional
        }
        bool ok = true;
        for (int j=1; j<=bugs; ++j) % For all bugs
            if (color[j] == 0 && !search(j, 1)) {
                ok = false; % If it fails even once, it is inconsistent
                break;
            }
        if (t)
            printf("\n");
        printf("Scenario␣#%d:\n", t+1);
        if (! ok)
            printf("Suspicious␣bugs␣found!\n");
        else
            printf("No␣suspicious␣bugs␣found!\n");
    }
}
```

As noted in the problem statement, `cin` in poj is more than 10 times slower than `scanf`, so use `scanf`.

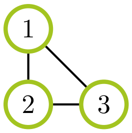
Graph Traversal To solve Bug's life, it is sufficient to ”traverse all edges” and confirm that each vertex can be colored with 1 and -1 (the numbers of adjacent vertices are different). This can be solved with breadth-first search (BFS) or depth-first search (DFS), which are methods for ”traversing all edges” in a graph. Either is fine, but here we will try creating it with DFS.

Since the sample input in the problem statement is too small, let's check the operation with a slightly more complex graph.

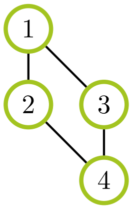


```
1
7 6
1 2
1 4
1 5
2 3
5 6
5 7
```

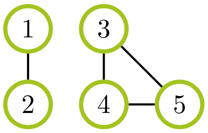
Checking for Merges/Loops In a general graph, there may be loops. The behavior that the program should take differs between odd and even loops, so create more than one example and check the operation.



```
1
3 3
1 2
1 3
2 3
```



```
1
4 4
1 2
1 3
3 4
2 4
```



```
1
5 4
1 2
3 4
4 5
3 5
```

1. Give the above data to the aforementioned BFS or DFS and confirm what kind of behavior (vertex visit order) occurs.

2. By appropriately adding to the `if (color[j] != 0) {....}` part in the above code example, make it return `false` in the case of inconsistency.

When submitting to poj, be sure to remove the output to `cerr` for operation confirmation.

8.12 Various Graph Searches

Problem **Curling 2.0**⋆ (National Preliminary 2006)

Find the minimum number of moves to reach the goal while sliding on the ice, or determine if it is impossible within 10 moves.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1144&lang=jp`

Since there is a limit of 10 moves, DFS is suitable. DFS uses memory proportional to the maximum depth and the number of sibling nodes. On the other hand, BFS uses memory proportional to the number of nodes at the same distance from the root in the BFS tree. Usually, the latter is larger.

This time, since the walls break as well as the puck's position changes, it is necessary to model both.

Problem **Articulation Points⋆** (AOJ)

Find the articulation points.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=GRL_3_A&lang=jp

A vertex that, when removed, makes the graph disconnected is called an articulation point. Along with bridges (edges that, when removed, make the graph disconnected), which will be discussed next, it is a concept that represents the strength of connectivity in a graph.

Articulation points can be found using DFS. If T is the depth-first search tree of a graph G, then articulation points have the following properties:

- If the root of T has multiple children, it is an articulation point.
- For a vertex n in T and its child c, if n.depth <= c.min holds, then n is an articulation point of G. Here, x.depth represents the depth of vertex x (the distance from the root in T), and x.min is the minimum value of:
  - x.depth
  - y.depth of any vertex y adjacent to x in G but not adjacent in T,
  - z.min of any descendant z of x in T

Specifically, perform a depth-first search while determining x.depth and x.min.

Problem **Bridges⋆** (AOJ)

Find the bridges.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=GRL_3_B&lang=jp

Problem **Map of Ninja House⋆** (Asian Regional 2002)

Reconstruct a map from a search history.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1236&lang=jp

Problem **Karakuri Doll**$\star$ (Mock National Preliminary 2007)

Verify the operation of a doll made by the Karakuri doll maker JAG.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2017&lang=jp`

Problem **Two Parties**$\star\star$ (12th Polish Olympiad in Informatics)

Can a graph be divided into two parts so that the degree of all vertices is even?

`https://szkopul.edu.pl/problemset/problem/eC-cABL-jWd4JdZDmfWufeeQ/site/`

Problem **Cakes**$\star\star$ (Algorithmic Engagements 2009)

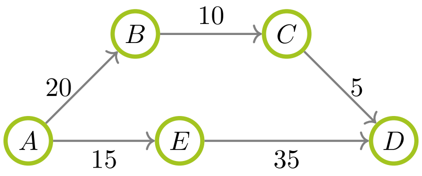
Since many pastry chefs have gathered, create as many teams of three as possible. The maximum amount of flour consumed per chef in a team is consumed by the team, and cakes are made for each team. What is the maximum amount of flour consumed in total by all teams? A chef can belong to multiple teams, but to form a team of three, all pairs of relationships within the team must be good.

`https://szkopul.edu.pl/problemset/problem/XWoCTR5RfPUYrlXIGPad9u2W/site/?key=statement`

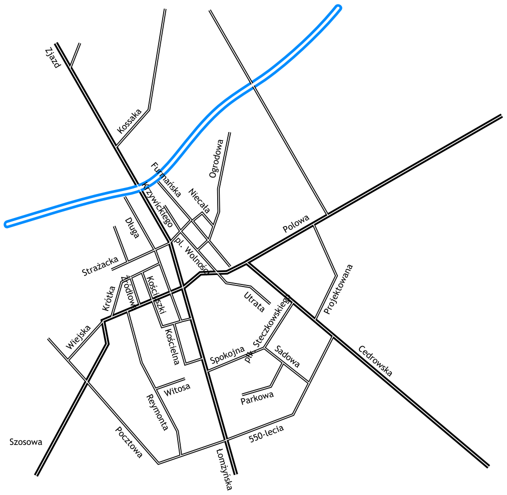
Chapter 9

Graphs (3) Shortest Path Problems

Example Problem



What is the cheapest way to get from A to D? A..E are towns. Roads connecting towns have a specified cost. ➔ The path {A,B,C,D} has the minimum cost of 35 (it is disadvantageous to move to E first).



9.1 Weighted Graphs and Representations

This section deals with shortest path problems on graphs where edges have weights. For example, if the nodes of a graph are cities, the edges are means of transportation, and the costs on the edges are travel times, then the shortest path problem corresponds to the problem of moving to a destination quickly. If the costs are tolls, then it corresponds to finding the cheapest way to reach the destination. In the case of a directed graph where edges have directions, one-way streets can be represented. Note that if there is no direction, it can be considered as a special case of a directed graph where the costs in the opposite direction are always equal.

First, we assume that the costs are non-negative (you may have to pay a toll, but you will not receive a reward), and we will consider the general case later. For the representation of graphs in the first half of this chapter, we will use the simplest **adjacency matrix** (8.7, Reference[3, pp. 90, 91]). The i, j element $K[i][j]$ of the adjacency matrix K represents the cost if there is a directed edge from i to j, and ∞ if there is no edge.

9.2 All-Pairs Shortest Paths

There are various algorithms for solving shortest path problems, but it is good to first memorize the Floyd-Warshall algorithm, which finds the shortest paths between all pairs of nodes (Reference[3, p. 97]). As you can see, it is a simple and easy-to-implement algorithm that just involves three nested `for` loops.

```
procedure FLOYD-WARSHALL(int K[][])    ▷ Calculates the shortest path cost $K[i][j]$ from $i$ to $j$ for all pairs
                                       ▷ Initial values are $K[i][j] = d_{ij}$ (if there is an edge between i and j)
                                       ▷ or $K[i][j] = \infty$ (if there is no edge)
    for $k = 1..N$ do                  ▷ Adjust as needed if city numbers are not $1..N$
        for $i = 1..N$ do              ▷ Be careful not to swap the index $k$ with $i$ or $j$!
            for $j = 1..N$ do
                if $K[i][j] > K[i][k] + K[k][j]$ then
                    $K[i][j] \leftarrow K[i][k] + K[k][j]$;    ▷ Update if going through $k$ is cheaper
```

The outline of the operation is as follows: In the initial state, $K[i][j]$ represents the travel cost when only directly connected edges are used. After the loop for $k = 1$ is completed, $K[i][j]$ represents the minimum value when moving from i to j (using directly connected edges) or moving in the order $i - 1 - j$. After the loop for $k = 2$ is completed, $K[i][j]$ represents the minimum value of the routes when moving from i to j or $i - 1 - j$ or $i - 2 - j$ or $i - 1 - 2 - j$ or $i - 2 - 1 - j$. In general, when the loop for $k = a$ is completed, $K[i][j]$ represents the minimum value of the cost of paths that can pass through $1..a$ as intermediate points.

Outline of Proof

Let D^a_{ij} denote the shortest path (one of them) from i to j that includes cities up to a as intermediate points. When $a \geq 2$, D^a_{ij} can be divided into cases where a is included and not included. If a is not included, it is a case where stopping at city a would be a detour, and it is the same as D^{a-1}_{ij}. If a is included, it is the case where we reach j via a from i. Here, it is not necessary to consider cases where we pass through a when moving from i to a or from a to j (it is sufficient to consider only those cases). (Since the cost of each edge is non-negative, it is sufficient to consider only paths that pass through each city at most once as candidates for the shortest path.) Therefore, the shortest paths for moving from i to a and from a to j are D^{a-1}_{ia} and D^{a-1}_{aj}, respectively. Combining the cases where we stop at a and where we do not, we get $D^a_{ij} = \min(D^{a-1}_{ij}, D^{a-1}_{ia} + D^{a-1}_{aj})$.

9.2.1 Example Problem

Example **A Reward for a Carpenter** (PC Koshien 2005)

A carpenter goes somewhere and returns. (See original text)
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=0117&lang=jp`

Input/Output The input this time is given separated by commas (,), not spaces. Using `scanf` makes it easier to read such data.

A note when using it in C++ is to include `cstdio` and, when using `scanf`, do not use `cin`.

C++

```
\begin{verbatim}
#include <iostream>
#include <cstdio>
using namespace std;
int N, M, A, B, C, D, x1, x2, y1, y2;
int main() {
    scanf("%d␣%d␣%d␣%d", &N, &M, &x1, &y1);
    for (int i=0; i<M; ++i) {
        scanf("%d,%d,%d,%d", &A, &B, &C, &D);
        cerr << "read␣" << A << '␣' << B << '␣' << C << '␣' << D
             << endl;
        // A -> B has cost C
        // B -> A has cost D
    }
}
\end{verbatim}
```

What is the Upper Limit? Since the maximum number of cities is 20, set the matrix K large enough. A point to note is that the city numbers are given from 1 to 20, and the beginning of a C++ array is [0]. This time, it is recommended to secure a larger array and use only the necessary parts (do not use [0]).

C++

```
\begin{verbatim}
int K[32][32];
\end{verbatim}
```

When implementing it as a program, it is necessary to use a finite number as ∞. This number needs to be (1) larger than any shortest path. The maximum value of the shortest path can be estimated by the product of the maximum cost of each edge and the number of edges when all edges are traversed. (2) It needs to be a number that is not too large so that it does not overflow even when doubled. (Because addition is performed in lines 5 and 6 in the procedure)

C++

```cpp
\begin{verbatim}
const int inf = 1001001001;
\end{verbatim}
```

In many cases, a value of about 1 billion is sufficient. (Verify that it does not exceed the estimate)

Initialization of the Adjacency Matrix Let's first implement the part that reads the input and sets the adjacency matrix. Then, create a function `void show()` to display the adjacency matrix and try displaying it. The display part can reuse the previous function. However, note that this time, we will not use column 0 and row 0.

Confirm that the following output is obtained for the sample input. (It is okay if a specific number is written instead of inf. Also, it is okay if the digits are not aligned, as long as you understand it.)

```
inf    2    4    4  inf  inf
  2  inf  inf  inf    3  inf
  3  inf  inf    4  inf    1
  2  inf    2  inf  inf    1
inf    2  inf  inf  inf    1
inf  inf    2    1    2  inf
```

Floyd-Warshall Next, implement Floyd-Warshall to calculate the shortest path cost. It is good to check how the matrix K changes each time the loop for the outermost `k` is performed.

After the first loop (`k=1`) is completed:

```
inf    2    4    4  inf  inf
  2    4    6    6    3  inf
  3    5    7    4  inf    1
  2    4    2    6  inf    1
inf    2  inf  inf  inf    1
inf  inf    2    1    2  inf
```

Final state:

```
  4    2    4    4    5    5
  2    4    6    5    3    4
  3    5    3    2    3    1
  2    4    2    2    3    1
  4    2    3    2    3    1
  3    4    2    1    2    2
```

Creating the Answer Now, the answer required by the problem is ”the carpenter’s reward,” which is ”the cost of the pillar” - ”the money the carpenter received from the lord” - ”the shortest cost from the carpenter’s town to the mountain village” - ”the shortest cost from the mountain village to the carpenter’s town”. Calculate and output the answer by referring to the matrix K and performing appropriate addition and subtraction.

Once accepted, try solving it with other methods as well.

9.2.2 Cases with Negative Edge Weights

Example **Shortest Path - All Pairs Shortest Path** (AOJ)

Find the shortest paths between all vertices. Note that negative weights are possible.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=GRL_1_C&lang=jp`

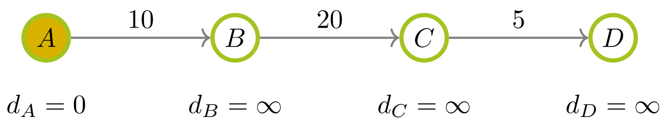
So far, we have only considered edges with non-negative weights, but there may be cases where it is appropriate to model edges with negative weights (negative edge). For example, consider a Sugoroku game where you receive a reward instead of paying a toll in certain sections. In such cases, how does the concept of the shortest path change? When costs can be negative, some of the properties that hold in the non-negative case do not hold, so caution is required. In particular, if there is a cycle where the sum of the costs is negative, a negative cycle, the shortest path cannot be defined because the cost continues to decrease as you keep going around the cycle. If there is no negative cycle on the path from the start to the end, the shortest path can be defined, and the minimum cost (although it may not be positive) is determined.

Fortunately, the Floyd-Warshall algorithm works even if there are negative cycles, and if $K[i][i] < 0$ for each point i at the end, then a cycle including point i exists. However, when implementing it, it is necessary to pay attention to the presence or absence of edges. For example, in the above procedure, add the condition that edges ik and kj exist to the part “`if` $K[i][j] > K[i][k] + K[k][j]$” on line 5. If the edge costs are non-negative, the presence of an edge could be confirmed by setting the cost of a non-existent edge to infinity, but if negative edges exist, the sum can become smaller even if one of them is infinity.

9.3 Single-Source Shortest Paths

The problem of finding the shortest paths from a single starting point to all other points can be solved more efficiently than finding the minimum distances between all pairs of points. For example, when calculating the round-trip cost, the solution can be obtained by solving the single-source shortest path problem twice, once for the outbound trip and once for the return trip.

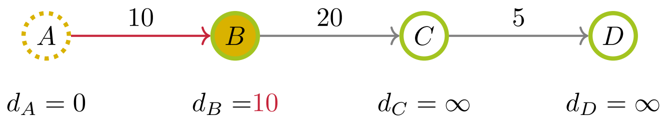
9.3.1 Relaxation



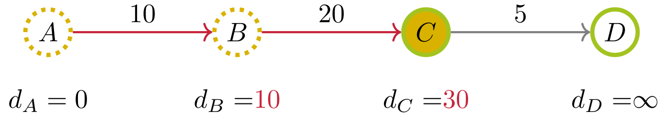
First, consider a simple graph like the one above and the problem of finding the distance from A to D, with A as the starting point.

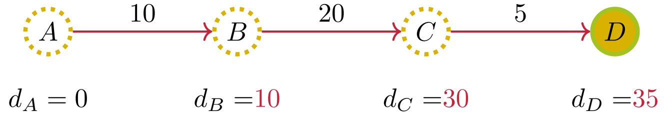
Definition: Let $d[x]$ be the upper bound of the shortest cost from A to x.

Initially, we define $d[A] = 0$ (there is no cost from A to A), $d[B] = d[C] = d[D] = \infty$ (since there is no information, it is ∞).



Definition: Relaxation Operation For a given edge (s, t) and its cost $w(s, t)$, if $d[t] > d[s] + w(s, t)$, then the operation of reducing $d[t]$ to $d[t] = d[s] + w(s, t)$ is called relaxation. If $\delta[t]$ is the true minimum distance to t, then $\delta[t] \leq \delta[s] + w(s, t)$. Therefore, if $\delta[n] \leq d[n]$ is maintained for all nodes, then this operation will also maintain $\delta[t] \leq d[t]$. Focusing on edge AB, $d[B] = \min(d[B], d[A] + 10) = 10$, and $d[B]$ changes from ∞ to 10.





Similarly, $d[C] = \min(d[C], d[B] + 20) = 30$, $d[D] = \min(d[D], d[C] + 5) = 35$, and the distance to D is obtained. By repeating the relaxation until d no longer changes, the true shortest cost is obtained. The efficiency differs depending on the order in which the relaxation is performed. Below, let V be the set of vertices, E be the set of directed edges, $w(u, v)$ be the weight of edge uv, and v_s be the starting point.

9.3.2 Bellman-Ford Algorithm

(Reference[3, p. 95])

```
procedure BELLMAN-FORD($V, E, w(u, v), v_s$)
    for $v \in V$ do
        $d[v] \leftarrow \infty$    ▷ Initialization: The upper bound of the distance to each vertex is $\infty$
    $d[v_s] \leftarrow 0$    ▷ Initialization: The distance to the starting vertex is 0
    for ($|V| - 1$) times do    ▷ It is also acceptable to repeat more than $|V|$ times
        for edge $(u, v) \in E$ do
            $d[v] \leftarrow \min(d[v], d[u] + w(u, v))$    ▷ Relaxation
```

- What is the order of relaxation? – Any order is fine.
- How long should it continue? Does it stop? – Repeat for all edges ($|V| - 1$) times.
- Is it really the shortest path? – Proven by the fact that the length of the shortest path is at most $|V| - 1$ (if there are no negative cycles).

9.3.3 Dijkstra's Algorithm

(Reference[3, p. 96])

```
procedure DIJKSTRA($V, E, w(u, v), v_s$)
    for $v \in V$ do
        $d[v] \leftarrow \infty$    ▷ Initialization: The upper bound of the distance from the starting point to each vertex is $\infty$
    $d[v_s] \leftarrow 0$    ▷ Initialization: The distance from the starting point to itself is 0
    $S \leftarrow \emptyset$    ▷ Set of vertices with confirmed shortest distances, initially empty
    $Q \leftarrow V$    ▷ Set of vertices with unconfirmed shortest distances, initially all vertices
    while $Q \neq \emptyset$ do    ▷ Repeat until there are no vertices with unconfirmed shortest distances
        select $u$ $s.t.$ $\arg\min_{u \in Q} d[u]$ ▷ Select $u$ from the "vertices with unconfirmed shortest distances" such that $d[u]$ is minimal
        $S \leftarrow S \cup \{u\}, Q \leftarrow Q \setminus \{u\}$    ▷ The minimum distance to $u$ is confirmed
        for $v \in Q$ $s.t.$ $(u, v) \in E$ do
            $d[v] \leftarrow \min(d[v], d[u] + w(u, v))$    ▷ Relaxation
```

- What is the order of relaxation? – For the vertices v that are destinations of edges from a vertex $u \in S$ with a confirmed shortest cost, select the vertex with the lowest cost (managed by linear search or priority queue, etc.).
- How long should it continue? Does it stop? – It stops when Q becomes empty.
- Is it really the shortest path? – Proven by contradiction (when w is non-negative).

9.3.4 Comparison of Methods and Negative Edges

Let V be the number of vertices. As can be seen by looking at the inner loop of the `for` statement, the Floyd-Warshall algorithm performs V^3 basic operations. If the time limit is about 1 second, it is manageable if $V = 100$, but it becomes difficult if $V = 1,000$. Using Big O notation, it is $O(V^3)$. Let E be the number of edges. The Bellman-Ford algorithm is $O(VE)$, and Dijkstra's algorithm is (depending on the implementation) about $O(V^2)$ or $O(E \log V)$, which is slightly more efficient. The number of edges E is about V^2 for a complete graph and about V for a graph close to a tree. Therefore, how much faster the Bellman-Ford and Dijkstra algorithms are than the Floyd-Warshall algorithm depends on the number of edges in the graph.

Example **Single Source Shortest Path I** (AOJ)

Find the shortest paths from city 0 to all other cities. The cities are numbered from 0 to $|V| - 1$. The input consists of the number of cities $|V|$ on the first line, followed by connection information from each node on the following $|V|$ lines.

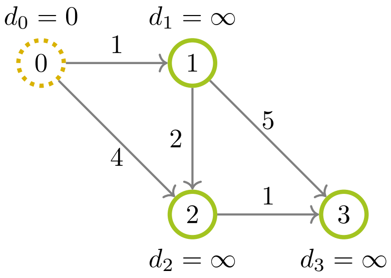
See the problem statement for details. Since the number of cities and edges is large, it is better to represent it with an **adjacency list** instead of an adjacency matrix.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=GRL_1_A&lang=jp`

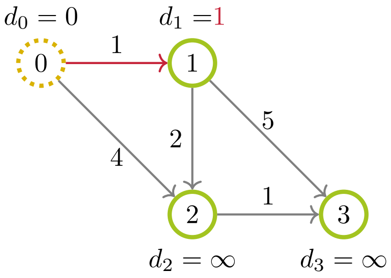
Graph Representation

In this problem, the number of cities is large, up to $100,000$. If we try to represent this with an adjacency matrix, we would need $(100,000)^2$ elements, which would definitely exceed the memory limit (try calculating it). On the other hand, the number of edges is at most $500,000$, which is much smaller than $(100,000)^2$, so it is better to manage them for each edge. In the case of the Bellman-Ford algorithm, it is sufficient to manage the start and end points as specified in the problem statement. When using Dijkstra's algorithm, it is convenient to use `vector` to create an adjacency list.

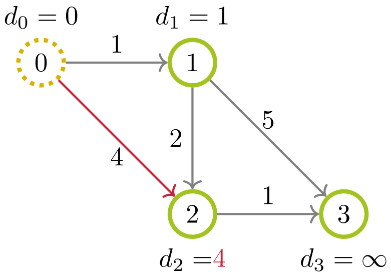
Let's try solving it with the Bellman-Ford algorithm For the graph of sample input 1, the operation when processing in the order of the given edges is as follows. (In this example, the shortest distance was found by looking at all the edges once by chance, but in general, it is necessary to repeat $|V| - 1$ times depending on the shape of the graph and the order of the edges.)



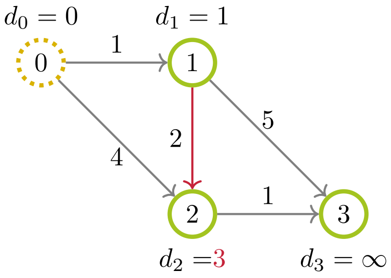
Initial state of sample input 1



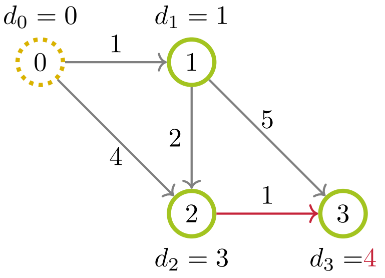
Relaxation with edge {0,1}: $d_0 + 1 = 1 < \infty$



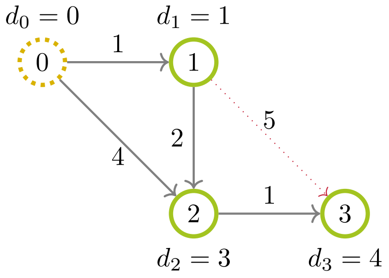
Relaxation with edge {0,2}: $d_0 + 4 = 4 < \infty$



Relaxation with edge {1,2}: $d_1 + 2 = 3 < 4$



Relaxation with edge {2,3}: $d_2 + 1 = 4 < \infty$



No update with edge {1,3}: $d_1 + 5 = 6 > 4$

C++

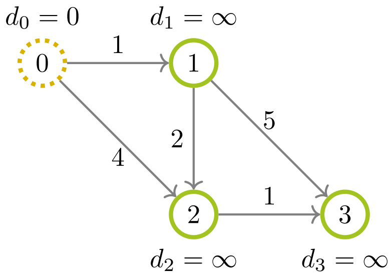
```
\begin{verbatim}
int V, E, R, S[500010], T[500010], D[500010]; // Input given in the problem
int C[100010]; // Upper bound of the shortest distance to each vertex
// Set a constant representing infinity to a value larger than the maximum path length
const int Inf = 10000*100000+100;

int main() {
  cin >> V >> E >> R;
  for (int i=0; i<E; ++i)
    cin >> S[i] >> T[i] >> D[i]; // Input each edge
  ... // Initialize C: Set C[R] to 0 and others to Inf
  for (int t=0; t<V; ++t) { // Repeat V times
    bool update = false;
    for (int i=0; i<E; ++i) {
      int s = S[i], t = T[i], d = D[i]; // For edge s,t,d
      if (C[s] < Inf && ...) { // If edge s,t can be relaxed
        C[t] = ...// Update C[t]
        update = true; // Record that an update occurred
      }
    }
    if (!update) break; // If there is no update after one loop, it is okay to stop
  }
  ... // Output
}
\end{verbatim}
```

Python3

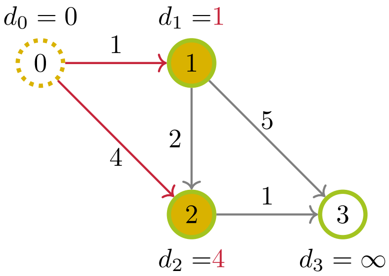
```
\begin{verbatim}
NV,NE,R = map(int,input().split()) # NX represents the number of X
Inf = 1001001001 # A value larger than the maximum path length
E = [tuple(map(int,input().split())) for _ in range(NE)]
# Initialization
D = [Inf for _ in range(NV)]
D[R] = 0
# Main loop
for t in range(NV):
    update = 0
    for s,t,d in E:
        # try to decrease D[t] w.r.t.  edge (s,t) with cost d, here
        # increment update if D[t] was changed (decreased)
    if update == 0:
        break
for v in range(NV):
    print(D[v] if D[v] != Inf else "INF")
\end{verbatim}
```

Let's try solving it with Dijkstra's algorithm When solving with Dijkstra's algorithm (Section 9.3.3), it is convenient to use a priority queue (see Chapter 7.2.3). Since C++ has `priority_queue` as a standard library, it is good to use it. Manage pairs of the distance from the starting point and the city number `pair<int,int>`, and in step 8, extract the city in order of increasing distance. For that purpose, in step 11, `push` the updated vertex into the priority queue. Originally, it would be efficient if the distance of the vertices inside the queue could be reduced, but it is difficult in many standard library implementations. Instead, `push` them redundantly, and ignore the vertices extracted for the second time or later. As a point to note, the **priority_queue** of the C++ standard library extracts elements in **descending order**, so it is necessary to specify a comparison function to change the behavior or to reverse the sign of the distance.

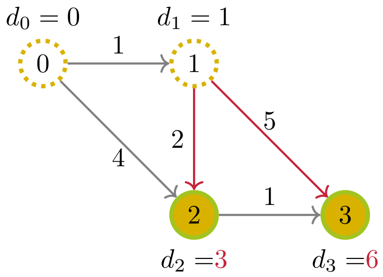
The following shows an example of the operation of Dijkstra's algorithm implemented with the above policy. The priority queue P has pairs of $\langle$distance from the starting point, vertex number$\rangle$ as elements, and the first element (left end) is extracted in order of increasing distance. Each graph in the figure corresponds to one execution of the loop from step 7.



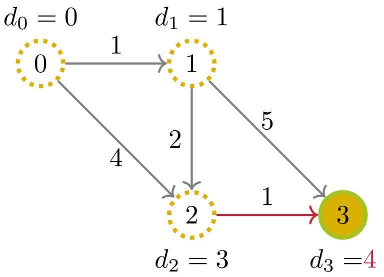
Initial state
$P = (\langle 0, 0\rangle)$



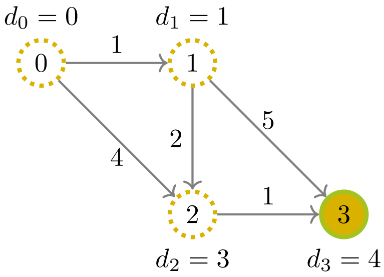
Extract $\langle 0, 0\rangle$, confirm vertex 0, relax vertices 1 and 2
$P = (\langle 1, 1\rangle, \langle 4, 2\rangle)$



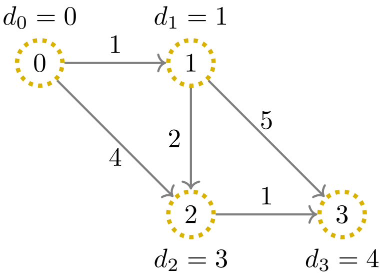
Confirm vertex 1 and relax vertices 2 and 3
$P = (\langle 3, 2\rangle, \langle 4, 2\rangle, \langle 6, 3\rangle)$



Confirm vertex 2 and relax vertex 3
$P = (\langle 4, 2\rangle, \langle 4, 3\rangle, \langle 6, 3\rangle)$



Ignore $\langle 4, 2\rangle$ since vertex 2 is already confirmed
$P = (\langle 4, 3\rangle, \langle 6, 3\rangle)$



Extract $\langle 4, 3\rangle$ and confirm vertex 3 with distance 4
$P = (\langle 6, 3\rangle)$

C++

```
\begin{verbatim}
#include <queue>
#include <vector>
#include <iostream>
using namespace std;
```

```
int V, E, R, S[500010], T[500010], D[500010];
int C[100010];
const int Inf = 10000*100000+100;
int main() {
    cin >> V >> E >> R;
    vector<pair<int,int>> edges[V];
    for (int i=0; i<E; ++i) {
        cin >> S[i] >> T[i] >> D[i];
        edges[S[i]].push_back(make_pair(T[i], D[i]));
    }
    for (int i=0; i<V; ++i) C[i] = Inf;
    C[R] = 0;
    priority_queue<pair<int,int>> Q;
    Q.push(make_pair(0,R));
    while (! Q.empty()) {
        pair<int,int> p = Q.top();
        Q.pop();
        int cur = p.second;
        if (C[cur] < -p.first) continue; // If it has already been visited, ignore it
        for (auto e : edges[cur]) {
            int dst = e.first, cost = e.second;
            if (C[dst] > C[cur] + cost) {
                C[dst] = C[cur] + cost;
                Q.push(make_pair(-C[dst], dst));
            }
        }
    }
    for (int i=0; i<V; ++i)
        if (C[i] == Inf) cout << "INF" << endl;
        else cout << C[i] << endl;
}
\end{verbatim}
```

Python3

```
import heapq
NV,NE,R = map(int,input().split()) # NX represents the count of X
Inf = 1001001001 # A value larger than any path that traverses all vertices
E = [tuple(map(int,input().split())) for _ in range(NE)]
# Initialization
D = [Inf for _ in range(NV)]
D[R] = 0
Q = []
```

```
heapq.heappush(Q,(0,R))
while len(Q) > 0:
    d,cur = heapq.heappop(Q)
    if D[cur] < d:
        continue
    for s,t,w in E:
        if s == cur and D[t] > D[s] + w:
            D[t] = D[s] + w
            heapq.heappush(Q,(D[t],t))
for v in range(NV):
    print(D[v] if D[v] != Inf else "INF")
```

Example **Single Source Shortest Path II** (AOJ)

Find the shortest paths from city 0 to all other cities when edges with negative weights are possible. See the problem statement for details.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=GRL_1_B&lang=jp

When there are edges with negative weights, Dijkstra's algorithm does not work correctly. The Bellman-Ford algorithm can find the correct solution if the shortest path is defined, and the existence of a negative cycle can be confirmed by checking whether an update is successful when trying to update for the $|V|$-th time.

Problem **Wormholes** (USACO 2006 December Gold)

Find a route that can return to a past time through wormholes. Any starting point is fine, as long as it is possible to return to a past time at that location.

http://poj.org/problem?id=3259

9.4 Exercises

Problem **Railway Connection**⋆ (National Preliminary 2012)

Find the route with the cheapest fare.
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1182&lang=jp`

Hint: There are two aspects, distance and fare, so handle them separately. (The rest is in white text)

Problem **Cliff Climbing**⋆ (National Preliminary 2007)

Find the shortest time to climb a cliff.
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1150&lang=jp`

Problem **Magical Dungeon**⋆ (Winter Camp 2008)

A hero with hit points H moves from room S and fights a monster the moment they arrive at room T. What is the maximum hit points they can have when they start the battle? The passages have positive and negative numbers assigned to them. If it is positive, the hit points are recovered, and if it is negative, they are lost. Movement that would cause the hit points to become 0 or less is not allowed. Movement that would cause the hit points to exceed H is possible, but the hit points can only recover up to a maximum of H.
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2124&lang=jp`
Problem and dataset: `http://acm-icpc.aitea.net/index.php?2007%2FPractice%2F%E5%86%AC%E5%90%88%E5%AE%BF%2F%E5%95%8F%E9%A1%8C%E6%96%87%E3%81%A8%E3%83%87%E3%83%BC%E3%82%BF%E3%82%BB%E3%83%83%E3%83%88` (day3, C)

Note: It is necessary to consider cases where you can

Problem **Broken Door**$\star$ (National Preliminary 2011)

Under the condition that a door somewhere is broken, find the shortest path considering the detour from the point where the broken door is found in the worst case.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1178&lang=jp

Problem **The Most Powerful Spell**$\star\star$ (National Preliminary 2010)

Find the lexicographically earliest spell that can be created.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1169&lang=jp

Problem **Sums**$\star$ (10th Polish Olympiad in Informatics)

Given a set of integers A. For a given query number, answer whether it can be represented as the sum of elements of A. (See the original text for the exact conditions)

Supplement: It is okay to assume that $a_0 \cdot n$ does not increase significantly up to the upper limit.

https://szkopul.edu.pl/problemset/problem/4CirgBfxbj9tIAS2C7DWCCd7/site/

Part II

Topics

Chapter 10

Plane Geometry

10.1 Overview: Representation and Operations of Points

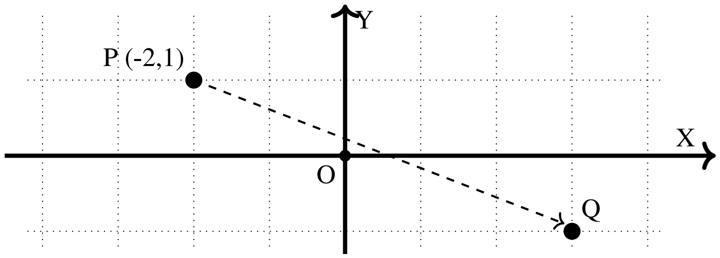
Overview

We encounter opportunities to handle geometric problems with computers in various situations. Here, I would like to introduce the basics of dealing with geometric problems (please take a specialized course to learn more). For example, let's try to represent familiar concepts such as parallelism and inside/outside of a figure using the tool of "signed area of a triangle." Even for simple geometric concepts for humans, there are cases where non-intuitive methods are suitable when describing them as programs. Also, since we are dealing with floating-point numbers (`double`, etc.), it is necessary to pay attention to errors.

When representing points on a plane in C or C++, for example, there is a method using structures (Reference$^{\text{Strategy}}$[2, pp. 365–(Chapter 16)]). In this material, to make it a little easier, we will represent them as complex numbers as follows.

10.1.1 Representation of Points Using Complex Numbers

Representation Using Complex Numbers in C++ In C++, the complex type in the standard library is used. (The real part `real()` corresponds to `x`, and the imaginary part `imag()` corresponds to `y`):



C++

```
#include <complex>
#include <cmath>
typedef complex<double> xy_t;
xy_t P(-2, 1), Q; // Initialization
cout << P << endl; // (for debug) display
cout << P.real() << endl; // x coordinate
cout << P.imag() << endl; // y coordinate
Q = P + xy_t(5, -2); // Point Q is the position of point P translated by (5,-2)
Q *= xy_t(cos(a), sin(a)); // Rotate point Q around the origin by a (radians)
cout << abs(P) << endl; // Length of vector OP
cout << norm(P) << endl; // norm(P) = abs(P)^2
```

Representation Using Complex Numbers in C Although the use of C is not recommended in this material, it is included here for the sake of the notes mentioned later.

In the case of C (gcc or C99):

C

```
#include <complex.h>
#include <math.h>
complex a = 0.0 + 1.0I; // Initialization
complex b = cos(3.14/4) + sin(3.14/4)*I;
printf("
a␣*=␣b;␣//␣Multiplication
printf("
```

Caution about Forgetting Multiplication Symbols

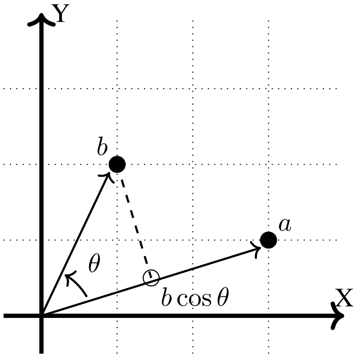
When calculating the product of the number 5 and the variable k, write 5*k; writing 5k will result in a compilation error. However, for the characters i, j, I, J, notations such as 5j are interpreted as the imaginary unit above and do not cause a compilation error. When displayed with cout, it is cast to bool and displayed as 1. This can be difficult to find if you are not aware of it.

Representation Using Complex Numbers in Python In Python3, complex numbers `complex` can be used in almost the same way. For details, please check `help(complex)` or `help(cmath)`.

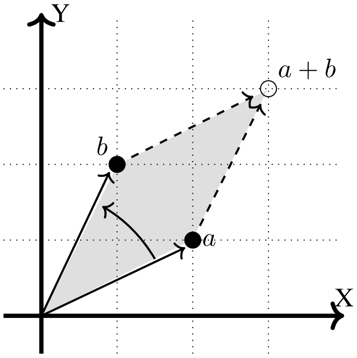
Python3

```python
import cmath
import math
P = complex(-2,1) # (-2+1j) is also acceptable
print(P)
print(P.real)       # Real part
print(P.imag)       # Imaginary part
Q = P + complex(5,-2)       # Translation
Q *= complex(math.cos(a), math.sin(a))      # Rotation
print(abs(P))       # Length
```

10.1.2 Frequently Used Operations



(1) Dot product: $|a||b|\cos\theta$



(2) Cross product: Signed area of the shaded part

C++

```cpp
// Figure (1) Dot product:  a.x*b.x +a.y*b.y
double dot_product(xy_t a, xy_t b) { return (conj(a)*b).real(); }
// Figure (2) Cross product, twice the signed area of the triangle formed by vectors a and b:  a.x*b.y - b.x*a.y
double cross_product(xy_t a, xy_t b) { return (conj(a)*b).imag(); }
// (No corresponding figure) Projection:  Project point p onto the line connecting the origin and b
xy_t projection(xy_t p, xy_t b) { return b*dot_product(p,b)/norm(b); }
```

The introduction of the signed areasigned area of a triangle is the main theme of the first half of this chapter. This calculates the area of the triangle formed by the origin and points a and b, with a sign. The sign is positive if the origin and points a and b are in a counter-clockwise positional relationship in this order, and negative if they are in a clockwise relationship. It is used not only for area but also for determining orientation, as we will see later.

The dot product is useful when projecting points onto a line.

Python3

```
def norm(c):
    a = abs(c)
    return a*a
def dot_product(a, b):
    return (a.conjugate()*b).real
def cross_product(a,b):
    return (a.conjugate()*b).imag
def projection(p, b):
    return b*dot_product(p,b)/norm(b)
```

10.2 Using the Signed Area of a Triangle

10.2.1 Area of a Polygon

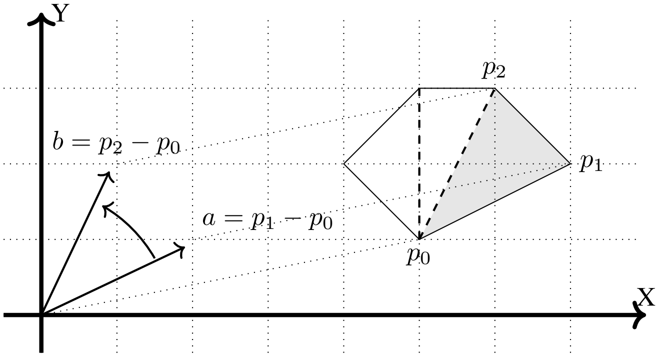
Example **Area of Polygon** (PC Koshien 2005)

Find the area of a convex polygon.

Note 1: Although Heron's formula is written in the problem statement, solve it using the signed area of a triangle below. (To apply it to the area of non-convex polygons later)

Note 2: The vertex sequence is given in order, but since it is not specified whether it is clockwise or counterclockwise, take the absolute value at the end.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=0079&lang=jp



Since a (simple) polygon can be decomposed into triangles, if the area of a triangle can be calculated, the area of a polygon can be calculated. Especially in the case of a convex polygon, it can be neatly divided by triangles formed by one vertex and the edges that do not include that vertex.

C++

```cpp
xy_t P[110];
int main() {
  // Input example:  Let N be the number of points read
  int N=0;
  double x, y;
  while (~scanf("%lf,%lf", &x, &y)) {
    P[N++] = xy_t(x,y);
  }
  // Area calculation
  double sum = 0.0;
  for (int i=0; i+2<N; ++i) {
    xy_t a=P[0], b=P[i+1], c=P[i+2];
    sum += ... // Add the area of triangle abc
  }
  printf("%.10f\n", abs(sum/2.0));
}
```

How to read as long as input continues with scanf

In order to read input lines as long as they can be read, the sample code adopts a loop of `while (~scanf("%lf,%lf", &x, &y))`. This is a short (but not very versatile) way and can only be used in an environment where `~EOF` becomes 0.

Python3

```
P = [] # Vertex sequence
try:
    while True:
        x,y = map(float,input().split(’,’))
        P.append(complex(x,y))
except EOFError:
    pass
N = len(P) # Number of vertices
total = 0.0
for i in range(1,N-1):
    a,b,c = P[0],P[i],P[i+1]
    total += cross_product(b-a,c-a) # Add the area of triangle abc
print("{:.10f}".format(abs(total/2.0)))
```

How to read as long as input continues with Python3

If `input()` is attempted and the input is finished, Python raises an exception called `EOFError`, so it is surrounded by `try` in advance and detected by `except`. When an exception occurs, the `while` loop where `input()` was written is exited, and the outer `except` block is executed (in this case, `pass`, which does nothing and moves to the next line).

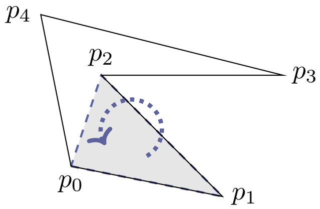
Example **Polygon - Area** (AOJ)

Calculate the area of a polygon. It is not necessarily convex, but the vertices are given in counterclockwise order.

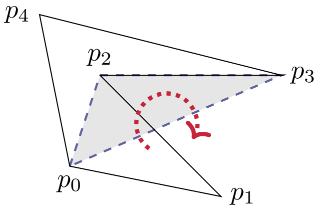
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=CGL_3_A&lang=jp

(Similar problem: Area of Polygons (Domestic Preliminary 1998) http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1100&lang=jp Only the direction in which the vertices are given is different)

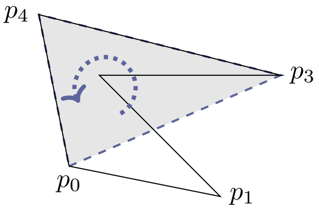
For a non-convex simple polygon, if the same division as before is performed, triangles will overlap, but if the sum is taken including the sign of the signed area, the correct area can be obtained (surprisingly). The vertices of the polygon must be given in counterclockwise order. Although p_0 was used as the center of the division, it is also possible to consider triangles formed by any point (for example, the origin) and each side.



$p_0p_1p_2$ (sgn: +)



$p_0p_2p_3$ (sgn: -)



$p_0p_3p_4$ (sgn: +)

10.2.2 Determining Parallelism

Example **Parallelism** (PC Koshien 2003)

Overview: Given four distinct coordinate points A = (x1, y1), B = (x2, y2), C = (x3, y3), and D = (x4, y4), determine whether the lines AB and CD are parallel.

The coordinates of the points are given as real numbers with up to 5 decimal places (use this information to estimate numerical errors).

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=0021&lang=jp

Solution Strategy: Determine whether the triangle formed by vectors AB and CD has an area.

C++

```cpp
const double eps = 1e-11;
double x[4], y[4];
int N;
int main() {
    cin >> N; // Number of problems
    for (int t=0; t<N; ++t) {
        for (int i=0; i<4; ++i)
            cin >> x[i] >> y[i]; // x0,y0..x3,y3
        xy_t a[2] = {
            xy_t(x[0],y[0]) - xy_t(x[1],y[1]),
            xy_t(x[2],y[2]) - xy_t(x[3],y[3])
        };
        bool p = abs(cross_product(a[0], a[1])) < eps;
        cout << (p ? "YES" : "NO") << endl;
    }
}
```

Note: For determining direction and angle, library functions such as `sin` and `arg` can be used, but from the viewpoint of calculation errors, it is better to calculate using the signed area as much as possible. For example, Taylor expansion is used in general implementations of trigonometric functions. [1]

Python3

```python
N = int(input())
for _ in range(N):
    P = list(map(float,input().split()))
    a, b, c, d = [complex(P[i*2],P[i*2+1]) for i in range(4)]
    parallel = abs(cross_product(b-a,d-c)) < 1e-11 # Calculate the boolean value indicating whether ab and cd are parallel
    print("YES" if parallel else "NO")
```

Operation Check

This problem has few sample input/output examples, and the judge data is also not public. Therefore, it is good to try your own test data. For example, long lines, short lines, and the positive/negative sign of the area are examples to check.

For example, the following examples are all "NO".

```
-0.00001 0 0.00001 0 -0.0001 0 0.00001 0.00001
-100 100 100 100 -100 100 100 99.99999
```

Handling Numerical Errors ⋆ When using floating-point numbers such as `double`, numbers that are not represented as the sum of powers of $\frac{1}{2}$ inevitably contain errors (see also Section B.4). In this problem, the input is explicitly stated to have an absolute value of 100 or less, and each value has a maximum of 5 decimal places. Therefore, the effect of errors can be avoided by multiplying by 10^5 and handling them as integers (`long long`). Alternatively, there is a method of predicting the range of errors, such as `eps` in the sample code. When errors are added to each element of two vectors (a, b) and (c, d), compare (1) the maximum value of $|ad - bc|$ when they are parallel (0 if there is no error) and (2) the minimum value of $|ad - bc|$ when they are not parallel, and set the threshold so that (1)<threshold< (2). Roughly estimating, (1) is at most $(4 \cdot 100) \cdot (100 \cdot 2^{-54}) \approx 2.2 \cdot 10^{-12}$ $(100 \cdot 2^{-54}$ is the representation error when a number up to 100 is represented by `double`, 400 is an estimate of the coefficient of ε when expanding $|(a+\varepsilon)(d+\varepsilon) - (b+\varepsilon)(c+\varepsilon)|$), and (2) is about 10^{-10} (from the square of the 10^{-5} value that can be represented by the input). Note that, depending on the environment, if you use a relatively new Intel or AMD CPU and a relatively new gcc, you can also perform calculations with better precision of 80-bit or 128-bit by using `long double` or `__float128`. There are points to note for each, so please investigate the literature if you use them.

[1] Reference: FreeBSD implementation https://svnweb.freebsd.org/base/release/10.1.0/lib/msun/src/k_cos.c?view=markup

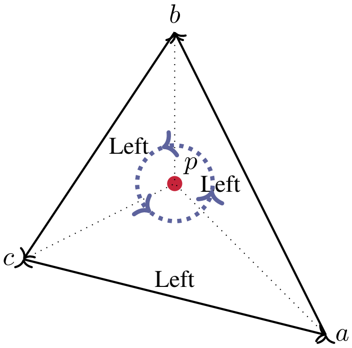
10.2.3 Inside/Outside Determination

Example **A Point in a Triangle** (PC Koshien 2003)

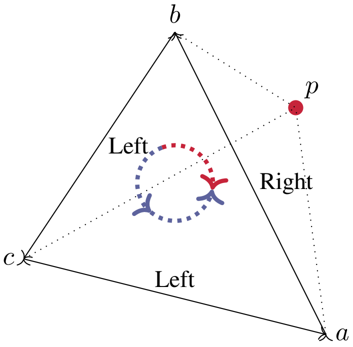
Given a triangle with vertices (x1, y1), (x2, y2), and (x3, y3) on a plane, and a point P(xp, yp), determine whether point P is inside the triangle (excluding the vertices and edges of the triangle).

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=0012&lang=jp

Example Solution: Let the vertices of the triangle be a, b, and c. Consider the signed areas of the three triangles pab, pbc, and pca. If p is inside abc, the signs will match; if it is outside, they will not match.



Point inside the figure



Point outside the figure

C++

```
double x[4], y[4];
int main() {
    while (true) {
        for (int i=0; i<4; ++i) cin >> x[i] >> y[i];
        if (!cin) break;
        xy_t a(x[0],y[0]), b(x[1],y[1]), c(x[2],y[2]), p(x[3],y[3]);
        // Twice the signed area of pab is cross_product(a-p,b-p)
        // Twice the signed area of pbc is cross_product(b-p,c-p)
        // Twice the signed area of pca is cross_product(c-p,a-p)
        bool ok = signs are the same
        cout << (ok ? "YES" : "NO") << endl;
    }
}
```

For inside/outside determination of polygons that are not necessarily convex, see the next section.

10.2.4 Convex Hull

Problem **Convex Polygon - Convex Hull⋆** (AOJ)

Find the convex hull of a given set of points. Refer to the diagram in the similar problem for what a convex hull is.

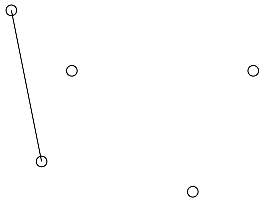
(Note: This is a slightly special setting that requires outputting points on the edges.)

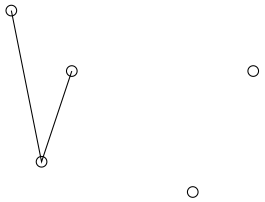
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=CGL_4_A&lang=jp`

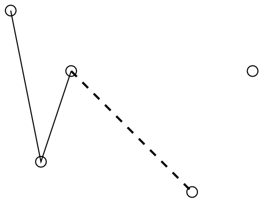
(Similar problem: Enclose Pins with a Rubber Band (PC Koshien 2004) `http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=0068&lang=jp` This one has a more straightforward setting)

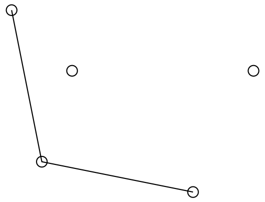
Example Solution: Sort the points by their X-coordinates, and starting from the minimum value (the leftmost point), go right, first constructing the lower half of the outer perimeter. If, during the process, the direction changes to the right of the current direction of travel (which means that a point that is not originally part of the convex hull has been included), remove all the points that caused this. Perform the same process for the upper half, starting from the rightmost point.

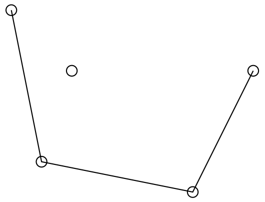
If the number of points is N, the above procedure for finding half of the perimeter can be done in $O(N)$, so the total computation time, including the time required for sorting, is $O(N \log N)$.


t=0: From the leftmost (minimum x-coordinate)


t=1: Connect the lines in order


t=2: If it turns right


t=4: Delete unnecessary points and reconnect


t=5: Lower side complete

Sorting Points Since the C++ complex number type (complex) does not have comparison operators defined, it is necessary to define them yourself. If you define `operator<` in the `std` namespace as in the example below, it will be automatically used by `sort`. As a comparison criterion, use something such that `a==b` if `!(a<b)` and `!(b<a)` are both true.[2]

C++

```
namespace std {
  bool operator<(xy_t l, xy_t r) {
    return (l.real()!=r.real()) ? l.real()<r.real() : l.imag()<r.imag();
  }
  // Alternative:  When aligning with std::pair
  bool operator<(xy_t l, xy_t r) {
    return make_pair(l.real(), l.imag()) < make_pair(r.real(), r.imag());
  }
}
```

10.3 Various Topics

Example **A Symmetric Point** (PC Koshien 2005)

Output the point that is symmetrical to point Q with respect to a line.
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=0081&lang=jp`

Example Solution: Let S be the point where Q is projected onto the line. Then, the desired point R is at the position where S is translated by the vector QS. For processing comma-separated input and controlling the number of decimal places in the output, it is convenient to use `scanf` and `printf` after `#include<cstdio>` as follows.

C++

```
#include <cstdio>
double X1,Y1,X2,Y2,XQ,YQ;

int main() {
  while (~scanf("%lf,%lf,%lf,%lf,%lf,%lf",
                &X1, &Y1, &X2, &Y2, &XQ, &YQ)) {
    xy_t P1(X1,Y1), P2(X2,Y2), Q(XQ,YQ);
    xy_t R = ...; // Calculate the point of line symmetry
    printf("%.10f,%.10f\n", R.real(), R.imag());
  }
}
```

[2] `https://en.cppreference.com/w/cpp/named_req/Compare`

Problem **Polygon - Polygon-Point Containment** (AOJ)

Determine whether a point is inside or outside a polygon that is not necessarily convex.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=CGL_3_C&lang=jp

(Note: Normally, it is difficult to determine whether a point is on an edge, but it is possible in this case because the coordinates are integers.)

Example Solution: Extend a half-line from the point you want to check in any direction and count the number of edges it crosses. If it is even, it is outside; if it is odd, it is inside. It is necessary to prepare functions in advance, such as whether a line segment has an intersection with a line. If the half-line passes very close to a vertex, it is safer to change the angle to avoid worrying about the effects of errors.

Problem **Point Set - Closest Pair**$\star$ (AOJ)

Find the closest pair of points.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=CGL_5_A&lang=jp

If the number of points is N, trying all pairs of points requires $O(N^2)$ time, but it is possible in $O(N \log N)$ using divide and conquer. There also exists a randomized algorithm with $O(N)$.

Divide and Conquer Strategy: Sort the points by X-coordinate and Y-coordinate, respectively. Divide the points in half by the X-coordinate, and recursively find the closest pair in the left and right halves. Let d be the smaller of the two minimum distances found. The closest pair of the whole is either the closest pair of the left half, the closest pair of the right half, or the closest pair of a point in the left half and a point in the right half. For the latter, when the points within a distance d from the dividing line between the left and right halves are sorted by Y-coordinate, it can be determined in a linear number of calculation steps with respect to the number of points by using the property that only pairs within a constant number (e.g., 8) are candidates. The proof is based on the fact that points cannot exist too densely with the restriction of d when considering an appropriate square grid around the dividing line. In the implementation, it is not possible to achieve $O(N \log N)$ if the points are sorted by Y-coordinate every time. It is better to sort the whole set once at the beginning and then distribute them during the division. Also, when dividing, it may be necessary to pay attention to the case where multiple points have the same Y-coordinate. In practice, this can be avoided by rotating the whole set randomly at the beginning.

10.4 Applied Problems

Problem **ConvexCut** (Summer Camp 2012)

Determine if there exists a point in a given figure such that it can be cut into two equal areas regardless of the cutting angle.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2442&lang=jp

It can be proven that this is only possible when the polygon has an even number of vertices and all pairs of edges that should be paired are parallel, or when the midpoints of the paired vertices are equal.

Example Answer (Input/Output)

C++

```
#include <complex>
#include <iostream>
#include <cstdio>
using namespace std;
int N, x, y;
typedef complex<double> xy_t;
xy_t P[60];
void solve() {
 ...
}
int main() {
    while (cin >> N) {
        for (int i=0; i<N; ++i) {
            cin >> x >> y;
            P[i] = xy_t(x,y);
        }
        solve();
    }
}
```

Example Answer (Midpoint Calculation)

C++

```
void solve() {
  // If it's odd, there is no such point
  ...
  xy_t a = (P[0]+P[N/2])*0.5; // Midpoint of P[0] and P[N/2]
  for (int i=1; i<N/2; ++i) {
    xy_t b = ... // Midpoint of P[i] and P[i+N/2]
    // If a and b do not match even with error considered, abs(a-b) > eps,
there is no such point
  }
  printf("YES\n");
}
```

Problem **Circle and Points**⋆ (National Preliminary 2004)

N points are given on the xy-plane. Move a circle of radius 1 on the xy-plane to enclose as many of these points as possible. At this time, answer the maximum number of points that can be enclosed simultaneously. Here, a circle ”encloses” a point if the point is inside or on the circumference of the circle. (This problem statement has sufficient descriptions regarding errors)

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1132&lang=jp

Since there are an infinite number of candidate circle positions, narrow down the candidates. (”Consider the limit” Reference[3, p. 229])

Way of Thinking

Let n be the maximum number of points that can be enclosed simultaneously, and assume there is a circle that encloses n points. If no point is touching the circle, even if you move it until one of the points inside touches, the number of points enclosed will not change. That is, it is sufficient to consider only circles that touch one of the points (even if you consider the remaining circles, the answer will not change). However, there are still an infinite number of such circles.

Assume there is a circle that encloses n points, and one point is on the circle. If you rotate the circle around that point

Caution for Exceptional Cases

The above idea is generally correct, but there are exceptional cases. That is,

the above property.

How to Count Points

Suppose you want to set the position of a circle that exactly covers a certain point p and check the number of points included in it. For that purpose, if you calculate the distance to the center of the circle for all points, you can generally determine it. However, for the point p itself, since it is located on the circle, it may be determined to be outside due to numerical errors. If you increase the tolerance here, there is a risk of determining points that are truly outside as inside. Therefore, the determination of point p should be treated specially, and

Example Answer:

C++

```cpp
int main() {
  // Initialize the maximum value
  for (/*point p*/) {
    for (/*point q*/) { // p!=q
      if (/*if there is a circle passing through pq*/) { // There can be 0, 1, or 2 cases
         /*Count the number of points inside while checking all points*/
         /*Update if it exceeds the maximum value*/
      }
    }
  }
}
```

Problem **Roll-A-Big-Ball** (National Preliminary 2008)

Find the maximum size of a large ball that satisfies the conditions.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1157&lang=jp

Problem **Space Golf** (Asian Tournament 2014)

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1348&lang=jp

Problem **Chain-Confined Path**⋆ (National Preliminary 2012)

Find the shortest path through a ring.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1183&lang=jp

Hint: Consider the shape of the shortest path.
Solution:

Problem **Neko's Treasure**⋆ (Mock Regional Preliminary 2009)

Find how many walls to overcome (refer to the problem statement).
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2181&lang=jp

Problem **Area of Polygons**⋆ (Asian Tournament 2003)

Find the area of the shaded part.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1242&lang=jp

Way of thinking: It's just a matter of slicing and summing, but there are quite a few points to note, such as when multiple lines pass through the same square.
Reference: http://www.ipsj.or.jp/07editj/promenade/4501.pdf

Problem **Treasure Hunt**⋆ (Summer Camp 2012)

Count the treasures in the area (efficiently).
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2426&lang=jp

If you count naively, it will take too much time, so perform preprocessing when the points are given and prepare to answer the questions. As a data structure for efficiently answering questions, for example, a quad tree can be used.

Problem **Altars**⋆⋆ (6th Polish Olympiad in Informatics)

A pillow that says that in China, evil spirits are believed to travel in straight lines.

There is a rectangular temple with an altar in the center. The walls of the temple are either east-west or north-south. The entrance to the temple is in the middle of one side. Check if there is a line of sight from the outside to the altar.

`http://main.edu.pl/en/archive/oi/6/olt`

Problem **Fish**⋆⋆ (Algorithmic Engagements 2009)

There are fish that wake up/go to sleep at the same time every day. When they wake up, they may be shifted by one square due to ocean currents. They swim so that they can see their position at the same time yesterday. They can accelerate and decelerate freely. There are many records of the fish's movements for one day, but the dates are unknown. What is the minimum number of fish that can be said to be observed?

`http://main.edu.pl/en/archive/pa/2009/ryb`

Chapter 11

Simple Parsing

Example Problems

Let's make a computer understand a string written in a specific grammar:

- $(V|V)\&F\&(F|V)$ ➔ F (Boolean calculation)
- 35=1?((2*(3*4))+(5+6)) ➔ '+' (Operator estimation)
- 4*x+2=19 ➔ x=4.25 (Solving equations)
- C2H5OH+3O2+3(SiO2) == 2CO2+3H2O+3SiO2 (Calculation of molecular weight)

11.1 Creating Arithmetic Operations

11.1.1 Let's Try Making Addition

Global Variables :

C++

```
const string S = "12+3";
size_t cur = 0; // Abbreviation for cursor, the parsing start position
int parse();
```

Execution Example: (We will create something that works as follows)

C++

```
int main() {
  int a = parse();
  assert(a == 15);
  assert(cur == S.size());
}
```

Python3

```
S = "0"
cur = 0

a = parse()
assert a == 15
assert cur == len(S)
```

Preparing a Function to Read One Character

C++

```
// Reads one character and advances cur
char readchar() {
  assert(cur < S.size());
  char ret = S[cur];
  cur += 1;
  return ret;
  // It can also be written in one line as return S[cur++];
}
// Reads one character but does not advance cur
char peek() {
  assert(cur < S.size());
  return S[cur];
}
```

In Python, to change a global variable within a function, specify it with `global`.

Python3

```
def readchar():
    global cur
    c = S[cur]
    cur += 1
    return c
def peek():
    return S[cur]
```

What is assert? (Re-posted)

C++

```cpp
#include <cassert>
int factorial(int n) {
  assert(n > 0); // (*)
  if (n == 1) return 1;
  return n * factorial(n-1);
}
```

Execution Example

C++

```cpp
cout << factorial(3) << endl; // Displays 6
cout << factorial(-3) << endl; // Stops by displaying the line number of (*)
```

First Addition

(Rough) Grammar of Addition

```
Expression := Number '+' Number
Number := Repetition of Digit
Digit := '0' | '1' | ... | '9'
```

How to read: (Reference: (Extended) BNF)

- `P := Q` ➔ Definition of a grammar rule named P
- `A B` ➔ B follows A
- `'a'` ➔ Character a
- `x | y` ➔ x or y

Implementing According to the Grammar (Digit)

```
Digit := '0' | '1' | ...  | '9'
```

C++

```cpp
#include <cctype>
int digit() {
  assert(isdigit(peek()));  // Check that S[cur] is a digit
  int n = readchar() - '0'; // Convert '0' to 0
  return n;
}
```

Conversion of characters to numbers: In C and C++, characters are managed by numbers representing those characters. Although the character code is not specified in the language standard, it can be considered as the ASCII code in the environment we currently use. In it, using the fact that characters such as ’0’, ’1’, ’2’, ..., ’a’, ’b’, ’c’ are assigned codes in order, the above subtraction shows how many characters after ’0’ it is, which is the numerical value we want. One way to check the ASCII code table is to use the man command, which is convenient. You can type `man ascii` in the terminal.

Python3

```python
def digit():
  assert peek().isdigit()
  n = int(readchar()) # Read one character and convert it to a number
  return n;
```

In the case of Python, it is good to determine with `isdigit()` as above and convert with `int`.

Implementing According to the Grammar (Number)

```
Number := Repetition of Digit
```

C++

```cpp
int number() {
  int n = digit();
  while (cur < S.size() && isdigit(peek())) // Look ahead one character to see if the next is also a digit
    n = n*10 + digit();
  return n;
}
```

Python3

```python
def number():
    n = digit()
    while cur < len(S) and peek().isdigit():
        n = n*10+digit()
    return n
```

Implementing According to the Grammar (Expression)

```
Expression := Number '+' Number
```

C++

```
int expression() {
  int a = number();
  char op = readchar();
  int b = number();
  assert(op == ’+’);
  return a + b;
}
```

This should work if it’s just addition:

C++

```
const string S = "12+3";
size_t cur = 0; // Parsing start position
..
int parse() { return expression(); }
int main() {
  int a = parse();
  cout << a << endl; // 15 should be output;
}
```

Test Try not only “12+5” but also “1023+888”, etc.

Extension: Add Subtraction

Let’s try “12-5” instead of “12+5”

Method: Determine if op is ’+’ or ’-’ in the expression function

C++

```
  if (op == ’+’) return a + b;
  else return a - b;
```

(Also rewrite assert appropriately)

Extension: Add Three or More

Let’s try “1+2+3+4” instead of “12+5”

Rewrite expression to allow multiple additions

C++

```
int expression() {
    int sum = number();
    while (cur < S.size() && (peek() == '+' || peek() == '-')) {
        // While addition or subtraction continues
        char op = readchar();
        int b = number();
        if (op == '+') add b to sum;
        else subtract b from sum;
    }
    return sum;
}
```

Next Extensions

- Support multiplication and division:
  Create new rules because the precedence of operators changes
- Support (multiple) parentheses:
  Create new rules and recurse

11.1.2 Arithmetic Operations Without Parentheses (Rough) Grammar

(Rough) Grammar of Arithmetic Operations

```
Expression := Term { ('+'|'-') Term }
Term := Number { ('*'|'/') Number }
Number := Digit { Digit }
```

How to read: (Reference: (Extended) BNF)

- `A B` ➔ B follows A
- `{C}` ➔ Zero or more repetitions of C

Example: 5*3-8/4-9

- Term: 5*3, 8/4, 9
- Number: 5, 3, 8, 4, 9

Implementation of Arithmetic Operations (Term)

```
Term := Number { ('*'|'/') Number }
```

C++

```cpp
int term() {
  int a = number();
  while (cur < S.size()
        && (peek() == '*' || peek() == '/')) {
    char op = readchar();
    int b = number();
    if (op == '*') a *= b; else a /= b;
  }
  return a;
}
```

In Python, integer division is possible with the `//` operator, but it is slightly different from the specifications of this problem. That is, in this problem, it seems necessary to evaluate `3/-2` as -1 instead of -2. Therefore, we use the `math.trunc` function.

Python3

```python
import math
def term():
    a = number()
    while cur < len(S) and (peek() == '*' or peek() == '/'):
        op = readchar()
        b = number()
        a = a*b if op == '*' else math.trunc(a/b)
    return a
```

Implementation of Arithmetic Operations (Expression)

```
Expression := Term { ('+'|'-') Term }
```

C++

```cpp
int expression() {
  int a = term();
  while (cur < S.size())
      && (peek() == '+' || peek() == '-')) {
    char op = readchar();
    int b = term();
    if (op == '+') a += b; else a -= b;
  }
  return a;
}
```

11.1.3 Arithmetic Operations: Introducing Parentheses

The entire expression, represented by Expression, appears again inside parentheses ➔ Process recursively

Grammar with Parentheses Introduced

```
Expression := Term { ('+'|'-') Term }
Term := Factor { ('*'|'/') Factor }
Factor := '(' Expression ')' | Number
```

An example implementation of `factor()` is as follows:

C++

```
int expression(); // Forward declaration
int factor() {
  if (peek() != '(') return number();
  readchar(); // Discard '('
  int n = expression();
  assert(peek() == ')');
  readchar(); // Discard ')'
  return n;
}
```

The implementation of `term()` should also be adjusted to match the grammar.

11.1.4 Summary

Summary of implementation:

- Create functions corresponding to the grammar rules
- The return value type should be what you want after parsing is complete
  - Arithmetic operations ➔ Integer
  - Polynomial ➔ Coefficients of each degree
  - Molecular formula ➔ Molecular weight, number of each atom...

Grammar description:

- Points to note: Operator precedence, left-associativity, and right-associativity
- Restriction: Be able to determine the appropriate rule with one character lookahead (LL(1))

Supplement Should the repetition of `P := A` { `'+' A` } be described recursively?

- `P := P '+' A | A`
  ➔ If implemented as is, P will recurse indefinitely
- If converted to right-associativity, parsing is possible
  - `P := A P'`
  - `P' := '+' A P' |` ϵ

  (ϵ is the empty string)

11.2 Practice Problems

Problem **Smart Calculator** (PC Koshien 2005)

Create a calculator.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=0109&lang=jp

Example Solution:

C++

```cpp
/*const*/ string S; // Removed const attribute because the value is changed
...
int main() {
    int N;
    cin >> N;
    for (int i=0; i<N; ++i) {
        cur = 0;
        cin >> S;
        S.resize(S.size()-1); // Ignore the last =
        cout << expression() << endl;
    }
}
```

Problem **Molecular Formula** (Asian Regional 2003)

Based on the atomic weights of each atom, find the molecular weight of each molecule.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1244&lang=jp

If you create a table at the beginning so that atoms can be converted to atomic weights, the rest is just multiplication and addition operations.

Problem **How Can I Satisfy You? Let's Count Them Up** (National Preliminary 2008)

Find the assignments of variables that satisfy a logical expression.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1155&lang=jp

Example Solution:

1. If there are no variables, it is easy to find the value of the expression. That is, it is sufficient to replace P, Q, and R inside the string with 0, 1, and 2, respectively, and then parse and calculate. Since there are 3^3 ways to assign values to P, Q, and R, repeat that many times.

2. (Especially recommended in C++11) Assume that the assignment of values to P, Q, and R is represented by an integer array `int a[3]`. Create a function by parsing that takes the assignment `a` as an argument and returns the value of the expression with that assignment. The part that utilizes the parsing result is as follows:

C++11

```
int solve() {
    cur = 0;
    auto tree = parse();
    int count = 0;
    for (int p:{0,1,2})
        for (int q:{0,1,2})
            for (int r:{0,1,2}) {
                int a[] = {p, q, r};
                if (tree(a) == 2) ++count;
            }
    return count;
}
```

The function that returns the value of the expression for an assignment is created by combining detailed functions. For example, if the character `c` represents a number, create a constant function that returns the same value regardless of the assignment: `return [=](int[3]){ return c-’0’; };`. If it is an alphabet, it returns a value according to the argument, so it should be `return [=](int a[3]){ return a[c-’P’]; };`. In the case of a binary operator enclosed in parentheses, after creating functions such as `left` and `right` that analyze the left and right sides in the same way as in the four arithmetic operations, create a function that calls the left and right functions like `return [=](int a[3]) { return min(left(a), right(a)); };`.

C++11

```
#include <functional>
typedef function<int(int[3])> node_t;
node_t parse() {
    char c = S[cur++];
    if (isdigit(c))
        return [=](int[3]){ return c-’0’; };
    if (isalpha(c))
        return [=](int a[3]){ return a[c-’P’]; };
    node_t left = parse();
    if (c == ’-’)  // Perform the ’-’ operation on left(a)
        return [=](int a[3]) { return ...; };
    assert(c == ’(’);
    char op = S[cur++];
    node_t right = parse();
    ++cur; // ’)’
    // Since the following is a binary operator, for left(a) and right(a)...
    if (op == ’*’)
        return [=](int a[3]) { return ...; }; // Perform the ’*’ operation
    return [=](int a[3]) { return ...; }; // Perform the same ’+’ operation
}
```

Problem **Equation Solver** (Ulm Local 1997)

Solve a simple equation.
http://poj.org/problem?id=2252

Example Solution: Parse the right and left sides respectively, and compare the coefficients of the first order and the constant terms on both sides.

Problem **Matrix Calculator** (Asian Regional 2010)

Let's do matrix multiplication.
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1314&lang=jp`

Problem **ASCII Expression** (Asian Regional 2011)

Calculate arithmetic expressions drawn in ASCII art style.
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1322&lang=jp`

Problem **Chemist's Math** (Asian Regional 2009)

Find the ratio of each substance required for a given reaction.
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1300&lang=jp`

Example Solution: Investigate the composition of each molecule and solve the system of equations.

Problem **Questions⋆⋆** (Algorithmic Engagements 2008)

Skillfully simulate the knowledge states of P princes and wizards, and guess how they would answer questions.
`http://main.edu.pl/en/archive/pa/2008/pyt`

Supplement

- Limitations: It is written that the maximum number of possible variable combinations is 600, and the absolute value of the variable values during calculation does not exceed 1 million, which are important points.
- In the sample input and explanation, it says "S 1 7 All sons know that there are less than 3 golden crowns.", but since there is "M 1 7" immediately after, this explanation is correct. Otherwise, the actual value of variable 7 cannot be read from "S 1 7".
- The person in charge's answer is about 160 lines.

Chapter 12

Repeated Squaring and Matrix Exponentiation

Overview

Experience how computation time can be shortened by using appropriate methods. In problems where this method is applicable, it is also possible to easily find the simulation result after 1 billion steps.
Applications:

- Find all the places you can go when a dice roll of $N(0 \leq N \leq 2^{31})$ occurs in a sugoroku game.
- Find the state of a biological colony that reproduces according to rules after N turns.
- Find the number of ways to paint according to rules.

(Reference[3, p. 114])

12.1 Approach

Let's consider the calculation of $3^8 = 6561$ as an example.

- $3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3$ ➔ 7 multiplications
- `int a = 3*3, b = a*a, c = b*b;` ➔ 3 multiplications

Exercise: What about 3^{128}? ➔ $\log(128)$ multiplications

Example **Elementary Number Theory - Power** (AOJ)

For two integers m and n, find the remainder of m^n when divided by $1\,000\,000\,007$.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=NTL_1_B&lang=jp

Related:

- Be careful of overflow: The range that can be represented by int is about 2 billion in this environment.
- Handling of remainders: (a*b)%M = ((a%M)*(b%M))%M

See Reference$^{\text{Strategy}}$[2, pp. 445–].

12.2 Language Features: struct and Recursive Functions

In this chapter, we will use `long long` and `struct`. Refer to Appendix B.3 and B.5 as needed.

12.3 Representation and Operations of Square Matrices

Note: The following is a sample code for a 2x2 matrix. You can create an NxN matrix yourself by using data structures such as arrays, vectors, or valarrays. If you need it for research or work, it is safer to use a dedicated library.

C++

```cpp
struct Matrix2x2 {
  int a, b, c, d; // a,b are the upper row, c,d are the lower row
};
```

Let’s also create a display function.

C++

```cpp
void show(Matrix2x2 A) {
    cout << "[␣" << endl
         << A.a << '␣' << A.b << endl
         << A.c << '␣' << A.d << endl
         << "]" << endl;
}
```

Multiplication of Matrices Next, we define multiplication. The following `mult` calculates the product of matrices `A, B`.

C++

```cpp
// returns C = A*B
Matrix2x2 mult(Matrix2x2 A, Matrix2x2 B) {
    Matrix2x2 C = {0}; // Initialize with 0
    C.a = A.a * B.a + A.b * B.c;
    C.b = A.a * B.b + A.b * B.d;
    C.c = A.c * B.a + A.d * B.c;
    C.d = A.c * B.b + A.d * B.d;
    return C;
}
// (Note) Be careful as exponentiation calculations can easily overflow: It is often necessary to add some tricks here.
```

Example of use:

C++

```cpp
  Matrix2x2 A = {0,1, 2,3}, B = {0, 1, 2, 0};
  show(A);
  show(B);
  Matrix2x2 C = mult(A, B);
  show(C);
```

Calculation of Exponentiation (Repeated Squaring) The following code calculates A^p (where $p > 0$) of matrix A and writes it to O.

C++

```cpp
// O = A^p
Matrix2x2 expt(Matrix2x2 A, int p) {
    if (p == 1) {
        return A;
    } else if (p % 2 == 1) {
        Matrix2x2 T = expt(A, p-1);
        return mult(A, T);
    } else {
        Matrix2x2 T = expt(A, p/2);
        return mult(T, T);
    }
}
```

Example of use:

C++

```
  Matrix2x2 A = {0,1, 2,3};
  Matrix2x2 C = expt(A, 3); // A to the power of 3
  show(C);
```

12.4 Practice: Fibonacci Numbers

Problem **Fibonacci** (Stanford Local 2006)

Calculate the remainder when the n-th value of the Fibonacci sequence is divided by 10^4.
$0 \leq n \leq 10^{16}$
http://poj.org/problem?id=3070

12.4.1 Various Calculation Strategies

Strategy 1: Calculate According to the Definition

C++

```
int fib(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fib(n-2)+fib(n-1);
}
```

Example of use:

C++

```
int main() {
  for (int i=1; i<1000; ++i)
    cout << "fib" << i << "␣=␣" << fib(i) << '\n';
}
```

Around n = 30, it stops progressing.

Strategy 2: Memorize Calculations Once Performed

Let's prepare an array called table and memorize the calculations once performed. This technique is called memoization (tabling) and has a wide range of applications.

C++

```
int table[2000]; // Memorize answers up to 2000
int fibmemo(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    if (table[n] == 0) // If it’s the first time
      table[n] = fibmemo(n-2)+fibmemo(n-1); // Calculate and memorize
    return table[n]; // Return the memorized value
}
```

Now, even for n=1000, the answer can be obtained immediately. (Although it overflows, we will ignore it for now)

However, with this method, it is only possible to calculate up to the number of elements in the `table` array. In the problem, the maximum value of n is 10^{16}, so it is not realistic.

Strategy 3: Calculate from the Smallest

It can be calculated in time proportional to n. With this method, if you wait, it will finish around 10^9, but 10^{16} takes too long.

C++

```
int fibl(int n) {
    int a[2] = {0,1};
    for (int i=2; i<=n; ++i) {
        // Invariant:  At the start of the loop, a[0] and a[1] are respectively
        // Fib(i-1) and Fib(i-2) (if i is odd)
        // Fib(i-2) and Fib(i-1) (if i is even)
        a[i%2] = a[0] + a[1];
    }
    return a[n%2];
}
```

12.4.2 Representation with Matrices

$$\begin{pmatrix} F_{n+2} \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix}$$

If we let $A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$, then from the associative property, we obtain:

$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = A^n \begin{pmatrix} F_1 \\ F_0 \end{pmatrix} = A^n \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

With this approach, even if n becomes as large as 10^{16}, it can be calculated realistically. However, since n exceeds the range that can be represented by `int`, use `long long` for the arguments of `expt()`, etc. Also, if calculated normally, the values of the elements will also exceed the range that can be represented by `int`, so use properties such as:

- (a*b)%M = ((a%M)*(b%M))%M

- (a+b)%M = ((a%M)+(b%M))%M

to keep them within a small range.

For operation verification, it is good to confirm that the answers match by comparing with the method of Strategy 3 for small n. The remainders modulo 10^4 are, for example, $F_{10} = 55$, $F_{30} = 2040$, etc.

12.5 Applied Problems

Problem **One-Dimensional Cellular Automaton** (Asian Regional 2012)

(Literal Translation) Raise a matrix to the power of T $(0 \leq T \leq 10^9)$
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1327&lang=jp`

Example implementation of an NxN matrix:

C++

```cpp
struct Matrix {
    valarray<int> a;
    Matrix() : a(N*N) { a=0; }
};
Matrix multiply(const Matrix& A, const Matrix& B) {
    Matrix C;
    for (int i=0; i<N; ++i)
        for (int j=0; j<N; ++j)
            C.a[i*N+j] = (A.a[slice(i*N,N,1)]*B.a[slice(j,N,N)]).sum();
    return C;
}
```

Problem	**Can You Go?**⋆ (UTPC 2008)

A sugoroku game with a huge die (sum of eyes up to 2^{31})
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2107&lang=jp`

- Expression of U-turn prohibition: Assign numbers to directed edges (path + direction), and create a transition matrix that represents which number (representing an edge) can be reached next from a certain number (representing an edge).
- The places that can be reached after n turns correspond to the product of the transition matrix raised to the power of n and the vector representing the initial position.
- It is recommended to return after reading Chapter **??** for graph representations such as adjacency matrices.

Problem	**Numbers**⋆ (GCJ 2008 Round1A C)

Summary: Find the last 3 digits of $(3+\sqrt{5})^n$
`http://code.google.com/codejam/contest/32016/dashboard#s=p2`

(Reference[3, p. 239])

How to Submit to GCJ

- Create a program: Check with Sample
- Click "Solve C-small" and download C-small-practice.in
- Create the output by redirection like `./a.out < C-small-practice.in > output.txt`
  Tip: If you use a command called `tee` like `./a.out < C-small-practice.in | tee output.txt`, it will output to the terminal while saving to a file, so you can see the progress.
- Submit `output.txt` from "Submit file". (The judgment will be displayed immediately)
- If it is correct, solve the large problem from "Solve C-large" as well.

Problem	**Leonard Numbers**⋆⋆ (POI Training Camps 2008)

Find the sum of Leonard numbers, which are similar to Fibonacci numbers.
`http://main.edu.pl/en/archive/ontak/2008/leo`

Chapter 13

Array Operations

13.1 Arrays, std::vector, std::array, and Operations

A brief introduction to methods for representing sequences of data and their operations.

13.1.1 Representation of Data Sequences

C++

```
  // Prepare an integer array of length 50 and initialize each element to 0 (valid from A[0] to A[49])
  int A[50] = {};
```

In C, when defining an array, it is necessary to give the length as a constant. When solving the problems handled in this material, estimate the required maximum value and take a little extra.

In the case of C++, the length can be determined at runtime by using `vector`.

C++

```
#include <vector>
using namespace std;
  int N = ...;
  // Prepare an integer array of length N and initialize each element to 0 (valid from A[0] to A[N-1])
  vector<int> A(N, 0);
```

Python3

```
n = 50
a = [0 for _ in range(n)]
```

In C++'s new standard, C++11, a data type called `array` has been introduced. This has an interface similar to `vector`, but it has an intermediate property between arrays and vectors in that the length must be given as a constant when defining it.

C++11

```
#include <array>
using namespace std;
  // Prepare an integer array of length 50 and initialize each element to 0 (valid from A[0] to A[49])
  array<int,50> A = {};
```

When using features introduced in C++11, in this seminar's iMac environment, use `g++-mp-4.8` instead of `g++`, and give the option `-std=c++11`. That is, when compiling `sample.cc`, it will be as follows.

```
$ g++-mp-4.8 -std=c++11 -Wall sample.cc
```

When submitting to AOJ, also select C++11 instead of C++. When writing in C++11, it is recommended to use `array` rather than raw arrays. There are advantages such as the type being preserved when received as a function argument, and out-of-bounds references can be detected by defining the macro `_GLIBCXX_DEBUG` in g++ 4.8.

C++11

```
#include <array>
int main() {
  std::array<int,3> a;
  a[4] = 5;
}
```

Accessing a[4] where it is up to a[2]

```
$ g++-mp-4.8 -std=c++11 -Wall sample.cc
$ ./a.out
# Nothing happens
$ g++-mp-4.8 -D_GLIBCXX_DEBUG -std=c++11 -Wall sample.cc
$ ./a.out
/.../c++/debug/array:152:error: attempt to subscript container
    with out-of-bounds index 4, but container only holds 3
    elements.
Objects involved in the operation:
sequence "this" @ 0x0x7fff6edfab00 {
  type = NSt7__debug5arrayIiLm3EEE;
}
Abort trap: 6
```

Although the message is difficult to read, anomaly detection was successful

13.1.2 Operations on Sequences

Various operations are available for such sequences. The specific operations available depend on the language.

Processing All Elements

When processing all elements of an array, it is common to use a for loop. Be careful not to make mistakes with the number of elements (5 in this case).

C++

```cpp
int A[5] = {0,1,2,3,4};
for (int i=0; i<5; ++i)
  printf("
```

In Python's for loop, there is no need to use indices.

Python3

```python
A = [3,1,4,1,5]
for e in A:
  print(e)
```

The same can be done in C++11.

C++11

```cpp
for (auto e:A) cout << e << endl;
```

Sorting: sort

See Section 4.1.2.

Reversing: reverse

C++

```cpp
#include <algorithm>
using namespace std;
int A[5] = {3,5,1,2,4};
int main() {
   sort(A,A+5);
   reverse(A,A+5); // Reverses the order of elements in the given range
   ...  // Try outputting A with cout
}
```

Python3

```python
A = [3,1,4,1,5]
b = list(reversed(A)) # Get b which is the reversed version of A (A remains unchanged)
A.reverse()    # Modifies A itself
```

Rotating: rotate

In C++, `rotate(a,b,c)` swaps the range [a,b) with the range [b,c).

C++

```cpp
#include <algorithm>
int A[7] = {0,1,2,3,4,5,6};
int main() {
  rotate(A,A+3,A+7);
  // For vector, it's rotate(A.begin(),A.begin()+3,A.begin()+7);
  // Try outputting A ➔  3 4 5 6 0 1 2
}
```

In Python, `a[p:q]` can be used to refer to or replace elements from position p to q in list a.

Python3

```python
a = [0,1,2,3,4,5,6]
a[0:7] = a[3:7]+a[0:3]
print(a) # ➔  [3, 4, 5, 6, 0, 1, 2]
```

Enumerating Permutations

Another unusual function is `next_permutation` in C++. A typical usage is to combine it with a `do .. while` loop to enumerate all permutations in ascending order, as shown below.

C++

```cpp
#include <algorithm>
    int A[4] = {1,1,2,3}; // Sort in ascending order beforehand
    // sort(A, A+4); // This is necessary depending on the initial values, but it's already sorted this time
    do {
        cout << A[0] << A[1] << A[2] << A[3] << endl;
    } while (next_permutation(A,A+4));
```

Check the output (are all cases actually covered?):

```
$ ./a.out
1123
1132
...
3121
3211
```

`next_permutation` rearranges the contents of the array and returns true if there is a combination that has not been tried yet (more precisely, the next element when all combinations are arranged in ascending order). Otherwise, it returns false. The loop ends when `next_permutation` returns false. (If you initialize the array with elements not sorted in ascending order (for example, `int A[4] = {2,1,1,3};`) and execute the above code, how will the output change?)

If you create this function yourself, it is expected that there will be many places where you can make mistakes, so test it thoroughly.

13.2 Practice Problems

Problem **Hanafuda Shuffle** (National Preliminary 2004)

Simulate a Hanafuda shuffle: Initially, there are N cards with values $1, \ldots, N$ arranged in a stack, with the bottom card being 1 and the top card being N. Find the top card after performing a shuffle defined by two parameters p and c as shown in the figure, R times.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1129&lang=jp

Input Example and Interpretation Interpreting the sample input

```
5 2
3 1
3 1
```

first, there are 5 cards, and the shuffle is performed twice. In the first shuffle, `[5, 4, 3, 2, 1]` becomes `[3, 5, 4, 2, 1]`, and in the second shuffle, it becomes `[4, 3, 5, 2, 1]`. Therefore, the top of the stack is 4.

Input and Program Structure First, create the input part and confirm that it is being read correctly by displaying n, p, and c. Next, represent the stack as a sequence of integers (array, vector, or array, etc.). It is good to implement the shuffle with rotation (rotate).

C++

```cpp
int N, R, p, c;
int main() {
    while (cin >> N >> R && N) {
        // Create the entire stack
        // Try displaying the created stack
        for (int i=0; i<R; ++i) {
            cin >> p >> c;
            // Perform shuffle p,c
            // Try displaying the entire stack for each shuffle
        }
        // Output the top of the stack
    }
}
```

Problem **Rummy** (UTPC2008)

A game using 9 cards. Determine whether the hand is in a winning state. A winning state is when the hand is in three ”sets” of three cards each. A set is a group of three cards of the same color, with the same number (1,1,1, etc.) or consecutive numbers (1,2,3, etc.).

The cards are in three colors: red, green, and blue, and the numbers are from 1 to 9.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2102&lang=jp

Input Example and Interpretation

```
1 1 1 3 4 5 6 6 6
R R B G G G R R R
```

3,4,5 and 6,6,6 are sets, but 1 has a different color ➔ Not a winning state

```
2 2 2 3 3 3 1 1 1
R G B R G B R G B
```

It is not a set to align the same numbers like 1,1,1, but it is possible to align consecutive numbers of 1,2,3 in the same color ➔ Winning state

Solution Policy When a human considers whether a hand is a match, it is imagined that they reach a final judgment with relatively few trials and errors by performing a clever (relatively complex) trial. Realizing ”how a human thinks” on a computer is often an interesting goal of artificial intelligence, but there are often other methods that are easiest for a computer. (Example: Looking at the sequence 122334, a skilled human can decompose it into 123, 234 at a glance, but can this operation be made into a strict rule without exceptions?)

Here, let’s create a solution with the policy of (1) listing all permutations of the cards, and (2) for each permutation, checking that the cards are in sets of three from the beginning. Such an approach of ”trying all possibilities with simple trials” is often suitable when solving problems with a computer.

Input Processing As usual, start by outputting the input as is. Let card[i] ($0 \leq i \leq 8$) represent the i-th card.

C++

```
#include <iostream>
#include <string>
using namespace std;
int T, card[16];
int main() {
    cin >> T;
    for (int t=0; t<T; ++t) {
        for (int i=0; i<9; ++i) {
          cin >> card[i];
          card[i]// Output
        }
        string color;
        for (int i=0; i<9; ++i) {
          cin >> color;
          color// Output
        }
    }
}
```

Color Conversion At the stage of processing the input, each card consists of a combination of two pieces of information, $\langle \text{color}, number \rangle$, but it is convenient for future operations to represent this with an integer. Therefore, let's represent the red cards [1,9] as [1,9], the green cards [1,9] as [11,19], and the blue cards [1,9] as [21,29], respectively. (Example: G3 is represented as 13, and B9 is represented as 29) [1]

C++

```
    cin >> T;
    for (int t=0; t<T; ++t) {
        for (int i=0; i<9; ++i) {
          cin >> card[i];
        }
        string color;
        for (int i=0; i<9; ++i) {
          cin >> color;
            if (color == "G") card[i] += 10;
            else if (color == "B") card[i] += 20;
          card[i] // Output and confirm
        }
    }
```

[1] This conversion only needs to ensure that all cards of different colors have different numerical values, so blue can be represented as [100,109], etc.

Set Determination Next, determine whether three cards form a set. As in the problem, there are two conditions for a set. Create and test each one separately.

C++

```cpp
bool is_good_set(int a, int b, int c) {
  return is_same_number(a, b, c) || is_sequence(a, b, c);
}
```

One condition is when the color and number are the same. In `card[i]`, different colors are represented by different integers, so it is sufficient to simply check if the numbers are the same.

C++

```cpp
bool is_same_number(int a, int b, int c) {
  // True if a, b, and c are the same, false otherwise
}
```

The other is when the color is the same and the numbers are consecutive. In `card[i]`, different colors are represented by different integers, and since there are no numbers 0 or 10, it is sufficient to simply check if the numbers are consecutive.

C++

```cpp
bool is_sequence(int a, int b, int c) {
  // True if a+2, b+1, and c are equal, false otherwise
}
```

Exercise: Consider why it is not necessary to consider a descending sequence of ”a-2, b-1, and c are equal”. (Can be postponed)

Test example:

C++

```cpp
int main() {
  // Test is_same_number
  cout << is\_same\_number(3, 4, 5) << endl; // False
  cout << is\_same\_number(3, 3, 3) << endl; // True
  // Test is_sequence
  cout << is\_sequence(3, 4, 5) << endl; // True
  cout << is\_sequence(3, 3, 3) << endl; // False
  // Test is_good_set
  cout << is\_good\_set(3, 4, 5) << endl;  // True
  cout << is\_good\_set(3, 3, 3) << endl;  // True
  cout << is\_good\_set(3, 3, 30) << endl; // False
  cout << is\_good\_set(5, 4, 3) << endl;  // False
}
```

Winning State Determination Determine whether the global variable card is in a winning state

C++

```cpp
bool is_all_good_set() {
    return // ((card[0],card[1],card[2] is a good set)
           // and (card[3],card[4],card[5] is a good set)
           // and (card[6],card[7],card[8] is a good set));
}
```

Overall Assembly Combining the code that creates all permutations and `is_all_good_set()` is almost complete.

C++

```cpp
int win() {
    // Sort card
    do {
       if // If this card permutation is a winning state
         return 1;
    } while (next_permutation(card, card+9));
    // All combinations have been tried, but it is not a winning state
    return 0;
}
```

After reading each data set, call this function `win()` and create a program that outputs the return value of 1 or 0. After testing locally, submit to AOJ and confirm that it is accepted.

13.3 Various Problems

Problem **Square Route** (Mock National Preliminary 2007)

In a city with approximately 1500 vertical and horizontal roads, count the number of squares.
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2015&lang=jp`

Note: It is not fast enough to list all four points and check if they form a square, so a clever method is needed.

Hint:

Problem **Multiplication** (Summer Camp 2012)

Determine how many steps a given procedure continues. (First, try to prove that it terminates in a finite number of steps, or create an example that continues infinitely.)

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2424&lang=jp

Problem **Starting Line** (Summer Camp 2011)

Run to the goal while accelerating with carrots.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2298&lang=jp

Problem **Water Tank⋆** (National Preliminary 2004)

There is a water tank with several partitions. Find the water level at a specified position and time.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1133&lang=jp

Problem **Ants⋆** (Polish Collegiate Programming Contest 2011)

Two ants are walking on a tree at a constant speed. Find the time they meet for the second time.

- One moves twice as fast as the other
- When going down a branch, they move twice as fast as when going up
- They start from the root in different directions
- When ants collide, they reverse direction
- When they reach the root, they reverse direction *after going around*
- The tree information is given by the sequence of up/down branches that one ant follows. (It's large, so it cannot be held in memory)

https://szkopul.edu.pl/problemset/problem/hKLVBaShgwgjH3R0nMREDw8g/site/

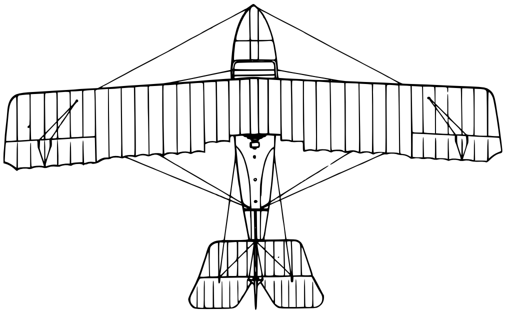
An account on main.edu.pl needs to be created separately from AOJ (only the first time). "Form/Year (a number)" might refer to the school year, since the English translation of the Polish version Klasa/Rok is Class/Year.

Problem **Pilot⋆** (17th Polish Olympiad in Informatics)

Calculate how straight a pilot can fly. The position (one-dimensional) at each time is given as a sequence a_i. The length of the sequence is at most 3,000,000. A range t is given as the allowed deviation. Find the maximum length of the range [i,j] in the sequence that satisfies the condition $|a_k - a_l| <= t, \; \forall k, l \; i \leq k, l \leq j$.

`https://szkopul.edu.pl/problemset/problem/lcU5m2RAICwNHsdzydb8JTQw/site/`

Note: Be careful as the values are large. Since there is a lot of input, `cout` is not fast enough (around 70 points), so use `scanf`. A solution with $O(n \log n)$ gets around 70 points.



Problem **Long and Narrow Place⋆⋆** (Summer Camp 2012)

Adjust the departure times of n carriages, and find the minimum time it takes for all carriages to reach the goal while satisfying the conditions.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2427&lang=jp`

Problem **Guesswork**⋆⋆ (11th Polish Olympiad in Informatics)

Since 9 numbers are given in order, answer what position it is in the whole sequence at the time each number is given. If all answers are correct when all 9 numbers have been answered, you win. Implement a strategy to maximize the winning rate (up to the point where you can get 100 points).

`http://main.edu.pl/en/archive/oi/11/zga`

(This problem's judge is currently not working)

Problems with high difficulty may be more appropriate to tackle after completing the first semester.

Chapter 14

Integers and Simultaneous Equations

14.1 Prime Factorization, Prime Numbers, Euclidean Algorithm, etc.

Example **Prime Factorize** (AOJ)

Prime factorize an integer.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=NTL_1_A&lang=jp`

It is good to try dividing by numbers in ascending order from the smallest. It is not necessary to divide by numbers larger than the square root of the dividend (why?).

Example **Greatest Common Divisor** (AOJ)

Find the greatest common divisor.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_1_B&lang=jp`

The **Euclidean algorithm** is a standard algorithm. See the explanation of the above problem, Reference$^{\text{Strategy}}$[2, pp. 441–443], and Reference[3, pp. 107–]. The least common multiple can also be easily found using the greatest common divisor (`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=NTL_1_C&lang=jp`).

Example **Sum of Prime Numbers** (PC Koshien 2004)

For a given number n, output the sum of the prime numbers up to the n-th prime number when the prime numbers are arranged in ascending order.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=0053&lang=jp`

The **Sieve of Eratosthenes** is a standard algorithm. First, prepare a large array to record whether a number is prime, and assume that all numbers are prime. Eliminate multiples of 2 except for 2, eliminate multiples of 3 except for 3, and eliminate multiples of 5 except for 5. Repeat the procedure of ”leaving the largest number that has not yet been eliminated as a prime number, and then eliminating its multiples.” For prime numbers larger than the square root of the largest prime number to be determined, it is not necessary to eliminate their multiples. See Reference$^{\text{Strategy}}$[2, pp. 438–439].

Problem **Galaxy Wide Web Service** (Summer Camp 2009)

There are periodic accesses to a web service from various planets. The time of one day on planet i is $1 \leq d_i \leq 24$ hours, and the current time is t_i o’clock. The amount of access from each planet is determined only by the time of that planet. (Fortunately, the time of any planet is in units of 1 hour on Earth.) Find the amount of access in the time zone with the most access.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2162&lang=jp`

Policy:

- If all periods can be found, it is roughly understandable. For planets with periods (1 2 3 4) and (2 1) as in the sample input,

Earth Time	0	1	2	3	4	5	6	7
Planet 1	1	2	3	4	1	2	3	4
Planet 2	2	1	2	1	2	1	2	1
Total	3	3	5	5	3	3	5	5

  The access count repeats every 4 hours, so the answer is 5.
- What is the total period? The least common multiple of the number of hours on the planets. If all times from 1 to 24 are present, it becomes 5354228880 (impossible).
- Treat 13, 17, 19, and 23 as special cases. The least common multiple of the remaining numbers is 55440. It is possible to enumerate at this time. The four specially treated numbers are relatively prime to all other periods, so the times when their maximum values overlap will eventually appear. In other words, it is good to find the maximum separately and take the sum of all of them.

Input and formatting: Summarize the access counts for each planet with a period of 1-24 hours (also absorb the time difference of t hours).

C++

```cpp
  while (cin >> N && N) {
    int Q[25][25] = {{0}}; // Q[d][i] amount at i-th hour of d-period (0<=i<d)
    for (int i=0; i<N; ++i) {
      cin >> d >> t;
      for (int j=0; j<d; ++j) {
        cin >> q;
        Q[d][(j+d-t)%d] = q;
      }
    }
    // Calculate the answer around here
  }
```

Ruby

```ruby
while true
 $N = gets.to_i
 break if $N == 0
 $Q = {}
 (1..$N).each {
  d, t, *q = gets.split("␣").map{|s| s.to_i}
  q.rotate!(t)
  unless $Q[d]
   $Q[d] = q
  else
   (0..d-1).each {|i| $Q[d][i] += q[i] }
  end
 }
 # Find the answer around here
end
```

With the processing up to this point, the access counts for that period are stored in Q[d][0] to Q[d][d-1] for $1 \leq d \leq 24$, so combine them as appropriate.

C++

```cpp
const int L = 16*9*5*7*11;
    int sum = 0, T[L] = {0};
    for (int d=1; d<=24; ++d) {
      if (d == 13 || d == 17 || d == 19 || d == 23 || d == 1)
        sum += /*(Maximum value from Q[d][0] to Q[d][d-1])*/;
```

```
            else
                for (int i=0; i<L; ++i) T[i] += Q[d][i%d];
        }
        cout << sum + /*(Maximum value from T[0] to T[L-1])*/ << endl;
```

Ruby

```
L = 16*9*5*7*11
  # Calculate the answer around here
  sum = 0
  $T = Array.new(L,0)
  $Q.each{|d,q|
    if d == 13 || d == 17 || d == 19 || d == 23 || d == 1
      sum += q.max
    else
      (0..L-1).each { |i| $T[i] += q[i%d] }
    end
  }
  # sum and $T.max are the answer
```

In this problem, the least common multiple was calculated manually in advance because there was a constraint of 1 to 24, but when calculating using a computer, it is good to use the Euclidean algorithm (Reference[3, pp. 107–]).

C++

```
int gcd(int a, int b) {
  if (a < b) swap(a,b);
  return b == 0 ? a : gcd(b, a%b);
}
```

Problem **Extended Euclidean Algorithm** (AOJ)

For two given integers a and b, find the *integer* solution (x, y) of $ax + by = \gcd(a, b)$.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=NTL_1_E&lang=jp

It can be found by the extended Euclidean algorithm. Utilize the information that can be found incidentally in the calculation performed by the Euclidean algorithm.

Modular Inverse

In problems with instructions such as ”find the remainder when the answer is divided by $M = 10^9 + 7$” in counting, etc., it is good to take the remainder from the intermediate stages using the relationships such as $(p+q)\%M = ((p\%M) + (q\%M))\%M$ and $(p \cdot q)\%M = ((p\%M) \cdot (q\%M))\%M$. This is because it can be calculated efficiently without using multi-precision integers, etc. Here, for division, it is necessary to convert $(p/q)\%M$ to $(p \cdot q^{-1})\%M$. q^{-1} is an integer that satisfies $(q \cdot q^{-1})\%M$, and is the modular inverse of q. When the integers q and M are relatively prime, that is, $\gcd(q, M) = 1$, q^{-1} can be obtained by the extended Euclidean algorithm for (q, M).

Problem **Chinese Remainder Theorem** (Classical)

”What is the number that leaves a remainder of 2 when divided by 3, a remainder of 3 when divided by 5, and a remainder of 2 when divided by 7?” In general, for remainders a', b', c' for mutually prime integers a, b, c, create a function that finds one ”number that leaves a remainder of a' when divided by a, a remainder of b' when divided by b, and a remainder of c' when divided by c”.

Since there does not seem to be a suitable problem on the online judge, please create input/output and test this problem yourself. Please include the tested data in the source code.

When performing calculations involving huge numbers, the computational cost is high if everything is calculated with multi-precision integers. Instead, it is more efficient to calculate the remainders for multiple prime numbers and restore them at the end. The number of legal positions in Go (19x19 board)[1] was obtained in this way.

14.2 Solving Simultaneous Equations

14.2.1 Language Features: valarray

When handling numerical vectors (such as rows or columns of a matrix), `valarray` is convenient.

C++

```
#include <valarray>
valarray<int> V(0, N);  // Creates an integer vector with N elements, each element is 0
V = 3; // Sets all elements to 3
V[0] = 1;
```

[1] 208168199381979984699478633344862770286522453884530548425639456820927419612738015378525648451698519643907 259916015628128546089888314427129715319317557736620397247064840935 (DOI:10.1007/978-3-319-50935-8_17)

```
V *= 10; // Multiplies all elements by 10
V

// Maximum value, minimum value, sum
cout << V.max() << ’␣’ << V.min() << ’␣’ << V.sum() << endl;

valarray<int> U(2, N);
V += U; // Sum of each element

// slice(initial, length, step)
V[slice(1,3,2)] = -3; // V[1] = V[3] = -3
```

14.2.2 Simultaneous Equations

Problem **Awkward Lights** (Asian Regional 2010)

Lights are arranged in a grid. When you press the switch of a certain light, the on/off state of the lights at a Manhattan distance d from that light changes, in addition to the light itself. Is it possible to turn off all the lights?

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1308&lang=jp

Approach: Since pressing a switch more than once is meaningless, each switch is either pressed or not pressed. Assign numbers from 0 to $MN - 1$ to each switch, and represent whether it is pressed by $x_i \in \{0, 1\}$. Let the matrix A be an $MN \times MN$ matrix, where the element a_{ij} indicates that pressing switch i affects light j. The product of matrix A and vector x is a vector representing the switches that have been changed. Let b_i represent whether each lamp was initially on, and find whether there is a solution to $Ax = b$ (where all operations are performed modulo 2).

If, after Gaussian elimination, the triangular part in the lower left of the matrix becomes 0, and no lights remain that cannot be turned off by any switch, then it is a success.

Input

C++

```
typedef valarray<int> array_t;
int M, N, D;
int main() {
    while (cin >> M >> N >> D && M) {
        valarray<array_t> A(array_t(0, M*N+1), M*N);
        for (int i=0; i<M*N; ++i) {
            cin >> A[i][M*N];
            for (int j=0; j<M*N; ++j) {
                int x0=i%N, y0=i/N;
                int x1=j%N, y1=j/N;
                int d=abs(x0-x1)+abs(y0-y1);
                if (d==0 || d==D) A[i][j] = 1;
            }
        }
        cout << solve(A) << endl;
    }
}
```

A is a matrix with MN rows and $MN + 1$ columns, where the rightmost column indicates whether the light is initially on, and the other elements indicate whether the j-th light is toggled when the i-th switch is pressed. $A[i]$ represents the i-th row, and each row corresponds to one `valarray<int>`. (Once you get used to it, it is more efficient to represent the entire matrix with a single `valarray<int>`)

Calculation

C++

```
\begin{lstlisting}[language=C++]
int solve(valarray<array_t>& A) {
    int p = 0;                  // top → bottom
    for (int i=0; i<M*N; ++i) { // left → right
        int r = -1;
        for(int k=p; k<M*N; ++k) if(A[k][i]) { r = k; break; } // "Find row where A[r][i]!=0 after position p"
        if (r==-1) continue;
        swap(A[p], A[r]);
        for (int j=p+1; j<M*N; ++j)
            if (A[j][i]) {
                A[j] = (A[j]+A[p])\%2;
                // Add row A[p] to row A[j], don't forget A[j]%=2
            }
        ++p;
    }
    for (int r=0; r<M*N; ++r) {
        bool ok = false;
        if(A[r][M*N]) {
            ok = true;
            for(int i=0; i<M*N; ++i) if(A[r][i]) ok = false;
        }
        if (ok) return 0;
    }
    return 1;
}
```

```
\end{lstlisting}
```

Problem **Strange Couple**⋆ (Summer Camp 2009)

A list of roads and intersections is given. At intersections without signs, people randomly choose a road with equal probability, including U-turns. At intersections with signs, people take the shortest route to the destination (if there are multiple routes, they choose randomly). Find the expected value of the total distance traveled from the starting point to the destination.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2171&lang=jp

Shortest path and simultaneous equations: First, find the distance from each intersection to the destination using Dijkstra's algorithm or similar. Then, using the expected value when starting from each intersection as a variable, set up simultaneous equations based on the relationship with the expected value when starting from adjacent intersections.

Since the method for finding the shortest path will be covered later, if you have not learned it yet, it is good to tackle it later.

14.3 Other Practice Problems

Problem **Afternoon Tea**⋆ (Polish Olympiad in Informatics 2011)

Which did you drink more of, tea or milk?

http://main.edu.pl/en/archive/amppz/2011/her

Problem **Making Change** (codechef)

The values of coins in a certain country are D[1], ..., D[N]. How many ways are there to pay an amount C (very large) with the coins of this country? However, D[1], ..., D[N] are distinct and any two are relatively prime.

http://www.codechef.com/problems/CHANGE

Chapter 15

Interpolation Polynomials and Numerical Integration

15.1 Lagrange Interpolation Polynomial

A quadratic polynomial passing through three points $(a, v_a), (b, v_b), (c, v_c)$ is uniquely determined as follows:

$$L(x) = \frac{(x-b)(x-c)}{(a-b)(a-c)}v_a + \frac{(x-a)(x-c)}{(b-a)(b-c)}v_b + \frac{(x-a)(x-b)}{(c-a)(c-b)}v_c. \tag{15.1}$$

In general, the lowest degree polynomial passing through N given points is uniquely determined:

$$L(x) = \sum_{k=0}^{N-1} \left(\prod_{i \neq k} \frac{(x - x_i)}{(x_k - x_i)} \right) \cdot v_{x_k}.$$

(If this formula makes you dizzy, you can skip it and proceed to Section 15.2.)

Problem **Find the Outlier** (Asian Regional 2012)

Given the degree d of a secret polynomial $f(x)$ and $d+3$ points, $(0, v_0), (1, v_1), \cdots, (d+2, v_{d+2})$. The point (i, v_i) satisfies $f(i) = v_i$, but as an exception, only one point $(i', v_{i'})$ contains an error of 1.0 or more in $v_{i'}$. Find that point i'. The input does not contain errors larger than 10^{-6}.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1328&lang=jp`

Strategy:

- Select $d{+}1$ points and create a Lagrange interpolation polynomial. Consider whether the value of that expression matches the unselected point.
- If we consider only the $d + 2$ points excluding i', any interpolation polynomial created from any $d + 1$ points will match f and pass through the remaining one point.
- For the $d{+}2$ points including point i', there is a way to select points ($d{+}1$ and 1) such that the same interpolation polynomial does not match the remaining one point.

The function `interpolate(n,E)` that obtains the point $f(n)$ from the $d{+}1$ points excluding the points (n, v_n) and (E, v_E) can be created as follows.

C++

```cpp
int D;
double V[16];
double interpolate(int n, int E) {
    // Calculate L(n) using the interpolation polynomial
    // However, do not use the points (n,V[n]) and (E,V[E])
    double sum = 0.0;
    for (int k=0; k<D+3; ++k) {
        if (/*if k is n or E*/) continue;
        double p = V[k];
        for (int i=0; i<D+3; ++i)
            if (! (/*if i is neither k nor n nor E*/))
                p *= (n-i)/(double)(k-i);
        sum += p;
    }
    return sum;
}
```

Python3

```python
# Assume values are in array V
def interpolate(n, e):
  total = 0.0
  for k in range(len(V)):
    if ... # if k is n or E
        continue
    p = V[k]
    for i in range(len(V)):
        if  ... # if i is neither k nor n nor E
            p *= 1.0*(n-i)/(k-i)
    total += p
  return total
```

Using this function `interpolate`, the function `outlier(E)` that checks whether there are no abnormalities in the remaining $d + 2$ points, assuming that the point (E, V_E) is abnormal, can be easily created as follows.

C++

```cpp
bool outlier(int E) {
    for (int i=0; i<D+3; ++i) {
        if (i==E) continue;
        double p = interpolate(i, E);
        if (/*if the absolute value (abs) of p and V[i] is more than 0.1 apart*/)
            return false;
    }
    return true;
}
```

Python3

```python
def outlier(e):
    for i in range(len(V)):
        if i == e:
            continue
        p = interpolate(i, e)
        if ... # if the absolute value (abs) of p and V[i] is more than 0.1 apart
            return False
    return True
```

C++

```cpp
#include<iostream>
#include<cmath>
// Define the above functions around here
int main() {
    while (std::cin >> D && D) {
        for (int i=0; i<D+3; ++i) std::cin >> V[i];
        for (int i=0; i<D+3; ++i)
            if (outlier(i)) { // If everything is consistent when ignoring i
                std::cout << i << std::endl;
                break;
            }
    }
}
```

Python3

```
# Define global variable V and the above functions beforehand
while True:
  d = int(input())
  if d == 0:
      break
  V = [float(input()) for _ in range(d+3)]
  for i in range(d+3):
      if outlier(i): # If everything is consistent when ignoring i
          print(i)
          break
```

15.2 Numerical Integration and Simpson's Rule

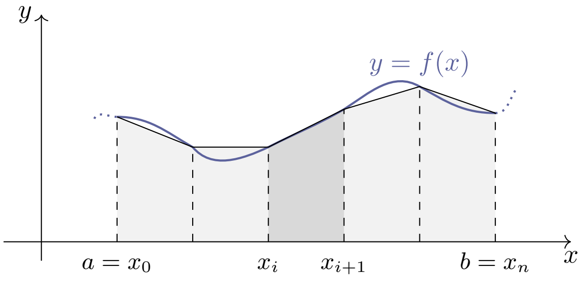
Example **Integral** (PC Koshien 2003)

Find an approximate value of the area using the sum of rectangles.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=0014&lang=jp

We want to find the definite integral $\int_a^b f(x)dx$ of a function $f(x)$ in the interval $[a, b]$. Besides analytical methods, there are various methods for finding approximate values by numerical calculation. The simplest method is to divide the interval finely and approximate it by the sum of the areas of rectangles. (The topic of the example problem ends here)

15.2.1 Trapezoidal Rule

For the divided strip portions, changing the rectangles to trapezoids can increase the accuracy corresponding to the number of divisions. If the division width is h, the error improves from $O(h)$ to $O(h^2)$ (if the effect of calculation errors is ignored, it can be expected that if the step size is made 10 times finer, the accuracy will improve 100 times).



Application

In reality, due to the effects of numerical errors, even if h is made smaller, the error will not decrease beyond a certain point. When the rounding error of each term is expected to be $\epsilon = 10^{-16}$, how should an appropriate h be estimated?

15.2.2 Simpson's Rule

In Simpson's rule, approximation is done with a quadratic equation instead of a straight line, and the accuracy is improved to $O(h^4)$. Although the notation is different from before, let $a = x_i,\ b = x_{i+1}$, and if we express the area of the interval using the interpolation polynomial (Equation (15.1)) using three points $(a, f(a))$, $(\frac{a+b}{2}, f(\frac{a+b}{2}))$, $(b, f(b))$ within the interval and simplify, the area of that interval can be approximated as follows:

$$\int_a^b f(x)dx \approx \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right).$$

If $f(x)$ is a polynomial of degree two or less within the interval, an accurate value can be obtained if calculation errors are ignored.

Problem **One** (Ritsumeikan Camp 2013)

Find the length of the boundary between the mountains and the sky visible within the window frame. Due to the input constraints, the peaks of the mountains exist within the window frame (including the boundary).

`http://judge.u-aizu.ac.jp/onlinejudge/cdescription.jsp?cid=RitsCamp13Day2&pid=I`

Note: The perimeter of $f(x)$ in the interval $x \in [a, b]$ is given by $\int_a^b \sqrt{1 + f'(x)^2} dx$, where $f'(x)$ is the derivative of $f(x)$.

Which parabola gives the boundary between the mountains and the sky varies depending on the location. Therefore, it is good to divide the x-axis into intervals and sum the perimeters for each interval. The division method is sufficient if we list the x-coordinates of the intersection points of each parabola (if any) and the intersection points with the lower edge of the window (if any), and use them to divide the inside of the window, and the number of intervals is also within a reasonable range. In each interval, either no mountain is visible at all, or one parabola forms the boundary with the sky. Which parabola is at the top can be found by comparing the heights at the center x of the interval. Numerical calculation is recommended for integration.

Problem **Intersection of Two Prisms⋆** (Asian Regional 2010)

Find the volume of the common part of a polygonal prism P1 with infinite height in the z-axis direction and a polygonal prism P2 with infinite height in the y-axis direction.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1313&lang=jp`

Strategy (Reference[3, pp. 236–]): Consider the cross-section of the desired solid cut by the plane $x = a$. Since the cross-section either has no area or is a rectangle parallel to the y-axis and z-axis, the volume is obtained by integrating the area in the x-axis direction. The intervals are divided by the x-coordinates where the vertices of the solid exist.

A function `width(polygon, x)` that finds the width of the y-axis or x-axis at a certain x-coordinate can be obtained by looping through the edges of the given polygon, for example, as follows.

Python3

```python
def width(polygon, x): # polygon contains the coordinates of the x-y or x-z plane in order
    w = []
    for i in range(len(polygon)):
        p, q = polygon[i], polygon[(i+1)%len(polygon)]
        if p[0] == x:
            w.append(p[1])
        elif (p[0] < x and x < q[0]) or (p[0] > x and x > q[0]):
            x0, y0 = p[0], p[1]
            x1, y1 = q[0], q[1]
            w.append(y0 + 1.0*(y1-y0)*(x-x0)/(x1-x0))
    assert len(w) > 0
    return max(w) - min(w)
```

Using this function, a function `volume` that finds the volume can be created, for example, as follows. Note that `$P1` is an array storing the (x,y) coordinates of polygon P1 in order, `$P2` is an array storing the (x,z) coordinates of polygon P2 in order, and `$X` is an array storing the x-coordinates appearing in P1 and P2 in ascending order.

Python3

```
def volume(Pxy,Pxz,X):
    X = sorted(set(X))
    total = 0.0
    xmin = max(min(Pxy)[0], min(Pxz)[0])
    xmax = min(max(Pxy)[0], max(Pxz)[0])
    for i in range(len(X)-1):
        a, b = X[i], X[i+1]
        if not (xmin <= a and a <= xmax and xmin <= b and b <= xmax):
            continue
        m = (a+b)/2.0
        va = width(Pxy, a)*width(Pxz, a)
        vb = width(Pxy, b)*width(Pxz, b)
        vm = width(Pxy, m)*width(Pxz, m)
        area = (b-a)/6.0*(va+4*vm+vb) # Definite integral of the interval [a,b] passing through (a,va), (m,vm), (b,vb)
        total += area
    return total
```

15.3 Fast Fourier Transform (FFT) as a Tool

(The explanation for this section has not been written yet)

Problem **Best Position**⋆ (Kuala Lumpur 2014)

Find the location that maximizes agricultural and livestock production. Field candidates are rectangular areas, and production occurs at matching cells when field patterns within them are superimposed.

https://icpcarchive.ecs.baylor.edu/index.php?option=com_onlinejudge&Itemid=8&category=645&page=show_problem&problem=4820

Represent each field candidate and field pattern as a polynomial, and devise a way so that a certain coefficient of the product polynomial becomes the sum of the areas when superimposed at a certain location.

Problem **Point Distance⋆** (Spring Contest 2013)

Each cell (x, y) is represented by C_{xy}, which indicates how many molecules are contained. Show the average distance and the histogram for each distance when considering all pairs of molecules.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2560&lang=jp`

Practice with 2D FFT. The histogram requires `long long`.

Problem **The Teacher's Side of Math⋆** (Asian Regional 2007)

Create a program that finds an integer coefficient polynomial where one of the solutions is $a^{1/m} + b^{1/n}$. (a, b are distinct prime numbers, m, n are integers greater than or equal to 2)

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1284&lang=jp`

Hint: In general, let ξ be a primitive m-th root of unity and η be a primitive n-th root of unity, then

$$\prod_{j=0}^{m-1}\prod_{k=0}^{n-1}(x - a^{1/m}\xi^j - b^{1/n}\eta^k)$$

By creating a polynomial, we can use the fact that it can be transformed as follows:

$$= \prod_{j=0}^{m-1}\{(x - a^{1/m}\xi^j)^n - b\} = \prod_{k=0}^{n-1}\{(x - b^{1/n}\eta^k)^m - a\}$$

Since the numbers are small, you can expand them appropriately, or use FFT.

Chapter 16

Sum/Maximum/Minimum of Intervals and Updates

Overview

Consider problems where, given a relatively large amount of data, you need to investigate a portion of the data for each query. By preparing in advance (≈ using an appropriate data structure), you can answer queries efficiently. Here, we consider data arranged in a row or on a grid.

16.1 Cumulative Sum

First, let's consider problems that utilize the sum from the beginning of a sequence a_i, $S_{i+1} = \sum_{k=0}^{i} a_k$. Let $S_0 = 0$.

Example **Maximum Sum Sequence** (PC Koshien 2003)

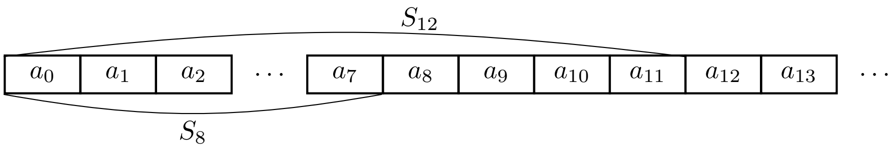
Given a sequence of integers $a_1, a_2, a_3, \ldots, a_n$, find the maximum sum of a contiguous subsequence of one or more terms.

Note: If a_i are non-negative, the sum of all terms is obviously the maximum, so it includes negative numbers. A subsequence including negative numbers may also have the maximum sum.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=0022&lang=jp`

(This is a problem called the maximum subarray problem, and there is an $O(N)$ solution where N is the length of the sequence. However, for the sake of the story, we will solve it by examining all intervals in $O(N^2)$.)

If we calculate the sum of the interval [i,j), that is, $\sum_{k=i}^{j-1} a_k$, according to the definition, it requires $j - i - 1$ additions. We want to complete this in one operation regardless of $j - i$. For that, it is sufficient to prepare an auxiliary variable S in advance such that $S_0 = 0$ and $S_{i+1} = \sum_{k=0}^{i} a_k = S_i + a_i$. Using this, the sum of the interval [i,j) can be found by $S_j - S_i$. For example, $a_8 + a_9 + a_{10} + a_{11} = S_{12} - S_8$.



Using this, we can obtain the maximum value by examining the sum of all intervals.

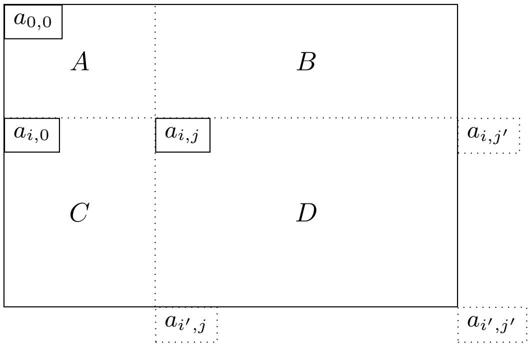
Problem **A Traveler** (JOI 2009)

Given a travel itinerary that goes back and forth, find the total distance walked (modulo).
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=0549&lang=jp

Idea: Suppose there is an instruction in the itinerary to ”move to the 30th post station to the left,” we want to avoid adding 30 times. Therefore, we calculate the distance from the left end for each post station in advance. Then, we can find the total distance with M additions, where M is the number of elements in the itinerary.

Two-Dimensional Cumulative Sum

The same idea can be applied to two dimensions: Define the auxiliary variable as $S_{i+1,j+1} = \sum_{k=0}^{i} \sum_{l=0}^{j} a_{k,l}$. The sum of the rectangular region with vertical [i,i') and horizontal [j,j') is $S_{i',j'} - S_{i,j'} - S_{i',j} + S_{i,j}$. (Subtract the extra from the large rectangle and adjust the over-subtracted part) In other words, it can be found with a constant number of operations regardless of the area of the rectangular region.



$$A = S_{i,j}$$
$$A + B = S_{i,j'}$$
$$A + C = S_{i',j}$$
$$A + B + C + D = S_{i',j'}$$

Example **Maximum Sum Sequence II** (PC Koshien 2003)

Find the rectangular region with the maximum sum.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=0098&lang=jp

(This problem can also be solved in $O(N^3)$ by converting it to a maximum subarray problem in either the vertical or horizontal direction. However, for the sake of the story, it is posted with the intention of solving it in $O(N^4)$ by examining all rectangular intervals.)

Problem **Planetary Exploration** (JOI 2010)

Find the amount of resources in a specified rectangular area.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=0560&lang=jp

Problem **Coffee Central**$\star$ (World Finals 2011)

Find the optimal location to open a cafe.
https://icpcarchive.ecs.baylor.edu/index.php?option=com_onlinejudge&Itemid=8&category=45&page=show_problem&problem=3133

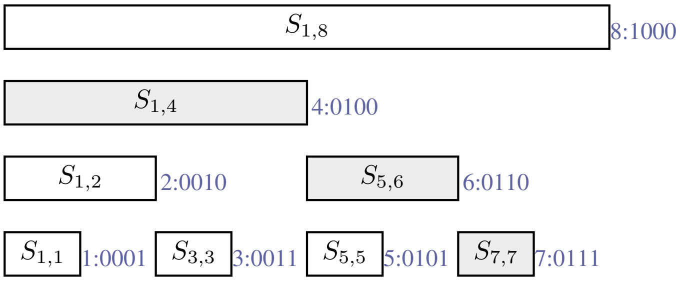
A diamond-shaped area may be easier to handle if rotated by 45 degrees.

16.2 Binary Indexed Tree (Fenwick Tree)

Now, let's consider a situation where some of the data a_i changes during processing. If we use the auxiliary variable S as in the previous section, we can answer queries in $O(1)$, which is efficient, but updates take $O(N)$. Here, we introduce a method that achieves $O(\log N)$ updates by allowing a cost of $O(\log N)$ for query responses. (Reference[3, pp. 160–])

Problem | **Range Query - Range Sum Query** (AOJ)

Manage the sum of values in a range.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=DSL_2_B&lang=jp



First, we calculate the sum for intervals with lengths that are powers of 2, as shown in the figure. Here, the notation $S_{i,j}$ represents the sum of the interval [i,j]. This notation differs from the previous sections, and includes j. Also, assume that the sequence to be managed starts from a_1 and 1. (It is also possible to start from 0, but fine-tuning is necessary as described later).

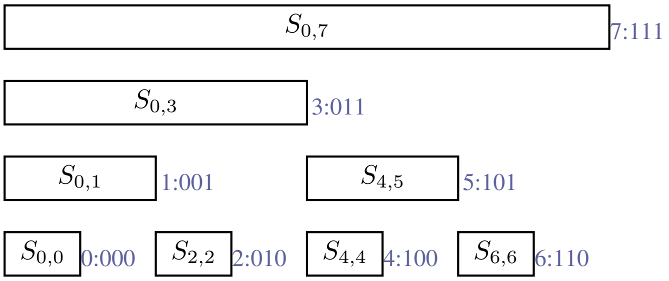
With such preparation, $S_{1,x}$ can be expressed as the sum of up to $\log x$ terms for $x \geq 1$. For example, $S_{1,7} = S_{1,4} + S_{5,6} + S_{7,7}$. Also, when the value of a certain element changes, it is sufficient to update at most $\log N$ partial sums. For example, when a_7 is updated, we update $S_{7,7}$ and $S_{1,7}$, which are the intervals that include it.

A data structure that manages such data in an array is called a Fenwick tree or Binary indexed tree (BIT). Using the element numbers (the right side of the colon is the binary representation) indicated in blue, we can manage it as follows. The element `BIT[n]` of the n-th array stores the sum $S_{l,r}$ of the interval where the left end l is the number obtained by dropping the lowest bit of n + 1, and the right end r is n. Therefore, the sum from 1 to n can be obtained by first adding `BIT[n]`, and then repeating the process of finding a new interval where the right end is $l - 1$ of the current interval until the left end l of the covered interval reaches 1. Such an interval can be found by dropping the lowest bit of `n`. Also, when updating a_n, after updating `BIT[n]`, we repeat the process of finding blocks that include that interval by expanding the right end (without reducing the left end).

C++

```
int BIT[1000010], bit_size;
void bit_init(int n) {
    fill(BIT, BIT+n, 0);
    bit_size = n;
}
int bit_sum(int n) { // Sum of [1,n]
  int ans = 0;
  while (n > 0) { // The left end of the entire interval contains only one 1, so it ends with 0
    ans += BIT[n];
    n &= n-1; // Drop the lowest (rightmost) 1:  Move to the left
  }
  return ans;
}
void bit_add(int n, int v) { // Add v to the n-th element
  while (n <= bit_size) {
    BIT[n] += v;
    n += n & (-n); // Carry up the lowest (rightmost) 1
  }
}
```

Reference: 0-index case



Sum: `n = (n & (n + 1)) - 1` $(n \geq 0)$ Carry down the 1 with 0 to the right

Update: `n |= n + 1` ($n <$ size) Set the rightmost 0 to 1

Problem **Moving** (Summer Camp 2012)

Find the total amount of energy required to sort in ascending order.
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2431&lang=jp`

Since sorting all elements is the sum from 1 to N, it is sufficient to subtract the maximum sum of the weights of the subsequences that do not need to be sorted (=ascending order).

If we denote the maximum sum of weights of ascending subsequences ending with the i-th number x_i as A_i, then

$$A_i = x_i + \max_{j<i,\, x_j<x_i} A_j$$

Therefore, it can be obtained sequentially by looking at the sequence from the left. Here, it takes time to check the maximum value for j smaller than i one by one, so we use a BIT with x_i as the key to process it efficiently.

C++

```cpp
long long N, x;
int main() {
    cin >> N;
    bit_init(N+1);
    for (int i=0; i<N; ++i) {
        cin >> x;
        long long cost = // Maximum value up to x
        ... // Update the maximum value up to x to cost+x
    }
    cout << (1+N)*N/2 - bit_max(N) << endl;
}
```

Ruby

```ruby
N = gets.to_i
X = gets.split('␣').map{|s| s.to_i}

tree = BIT_Max.new(N+1)
X.each {|x|
  cost = ... // Maximum value up to x
  ... // Update the maximum value up to x to cost+x
}
puts (1+N)*N/2 - tree.max(N)
```

Instead of a BIT that manages the sum of intervals, let's consider a data structure that has the maximum value of the interval $[1, n]$ from the beginning.

C++

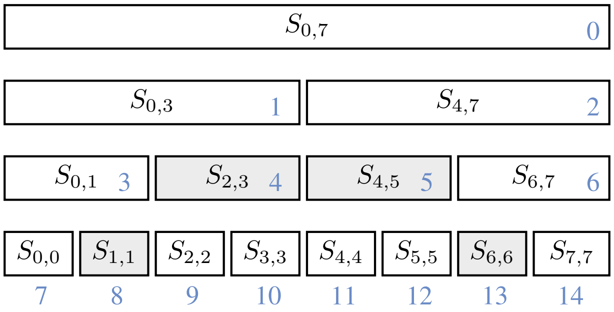
```
long long BIT[100010], bit_size; // long long is a 64-bit integer.
void bit_init(int n) {
    fill(BIT, BIT+n, 0);
    bit_size = n;
}
long long bit_max(int n) { // Maximum value of [1,n]
    long long ans = 0;
    while (n > 0) {
        ans = max(ans, BIT[n]);
        n &= n-1;
    }
    return ans;
}
void bit_setmax(int n, long long v) { // Update the maximum value of [1,n] to v
    while (n < bit_size) {
        BIT[n] = max(BIT[n], v);
        n += n & (-n);
    }
}
```

Ruby

```
class BIT_Max
  # 1-index
  def initialize(n)
    @array = Array.new(n+1, 0)
  end
  def max(n) # Maximum value of [1,n]
    ans = 0
    while n > 0
      ans = @array[n] if ans < @array[n]
      n &= n-1
    end
    ans
  end
  def setmax(n, v) # Update the maximum value of [1,n] to v
    while n < @array.size
      @array[n] = v if @array[n] < v
      n += n & (-n)
      end
    end
  end
```

16.3 Segment Tree and Range Minimum Query

Next, let's consider the minimum and maximum values of intervals. Unfortunately, the minimum value of the interval [i,j] cannot be obtained from the minimum values of the intervals [0,i] and [0,j]. Therefore, we will use a slightly richer data structure called a Segment Tree. (Reference[3, pp. 154–]).



We maintain the data for each interval in a complete binary tree with the entire interval as the root, numbered 0 as shown in the figure. Each interval other than the leaves has two children, with the left child responsible for the left half of the parent's interval and the right child responsible for the rest. For example, the minimum value of the interval [2,6] can be found by $\min(S_{1,2}, S_{2,3}, S_{4,5}, S_{6,6})$, as shown in the gray area of the figure. No matter what interval is taken, the number of places to check is at most 2 for each depth.

In this implementation, for a certain `id`, the left child is `id*2+1`, the right child is `id*2+2`, and the parent is `(id-1)/2`. If the length of the original array a_i is N, then if N is a power of 2, we use a region of $2N$, and if not, we use a region of at most $4N$.

Also, the least common ancestor (LCA) of a rooted tree can be calculated as an RMQ for the visit times of a depth-first search. Reference[3, pp. 292–]

Problem **Range Query - Range Minimum Query** (AOJ)

Manage the minimum value of an interval.
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=DSL_2_A&lang=jp

Problem **Balanced Lineup** (USACO 2007 January Silver)

Cows are lined up. What is the difference between the tallest and shortest cows in the interval [a,b]?

`http://poj.org/problem?id=3264`

scanf is necessary.

Problem **Drawing Lots**⋆ (UTPC2009)

For a lottery, find the minimum number of horizontal lines to add so that the selected location becomes a hit.

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2192&lang=jp`

Problem **Distance Queries**⋆ (USACO 2004 February)

Find the distance between two points on a farm.

`http://poj.org/problem?id=1986`

Problem **Apple Tree**⋆ (POJ Monthly–2007.08.05)

Count the number of apples.

`http://poj.org/problem?id=3321`

Chapter 17

Stable Matching (Stable Marriage Problem)

Consider the problem of matching elements between two groups, A and B. Examples include matching medical residents with hospitals, marriages between men and women, and products with factories. We will first address the problem where A and B have the same number of elements, N, and we create N pairs by selecting one element from A and one from B, ensuring no element is repeated (a **matching**[1]).

For example, if the theme is marriage, A and B would be men and women. Here, we assume that each individual has a list showing their preference ranking for potential partners, and the list includes all members of the opposite sex. In this setting, it is possible to create stable pairs that satisfy everyone's preferences in a certain sense. Here, stability means that there is *no* situation where ”a man and a woman would both prefer to break their current pairs and form a new pair.”

Examples of Stable and Unstable Pairs

Suppose that job types A and B are recruiting one person each, and applicants a and b have each determined their preference rankings. In the following examples, if Aa is not paired, then breaking the current pairs and forming the pair Aa would result in both A and a being in a more desirable situation than their current pairs.

Preference Ranking				Stable	Unstable	Reason for Instability
A	B	a	b			
ab	ba	AB	BA	(Aa, Bb)	(Ab, Ba)	First choices match
ab	ab	AB	BA	(Aa, Bb)	(Ab, Ba)	B prefers a but is unstable
ab	ab	AB	AB	(Aa, Bb)	(Ab, Ba)	B and b prefer each other but is unstable

This problem can be solved by repeatedly (1) determining temporary pairs and (2) replacing them with higher-priority pairs that are formed later (Gale–Shapley).

[1] In graph theory, a set of edges that are not adjacent to each other (i.e., **independent**) is called a **matching**.

C++

```
while not everyone is paired
  Select a single unmarried man, m
  Let f be the highest-ranked woman on m's␣preference␣list␣who␣has␣not␣yet␣rejected
␣␣if␣f␣is␣unmarried
␣␣␣␣Pair␣m␣and␣f
␣␣else␣if␣f␣is␣currently␣paired␣with␣m'
    if f prefers m'␣to␣m
␣␣␣␣␣␣m␣remains␣unmarried
␣␣␣␣else
␣␣␣␣␣␣Pair␣m␣and␣f
␣␣␣␣␣␣m' becomes unmarried
    end
  end
end
```

This problem can be solved by repeatedly (1) determining temporary pairs and (2) replacing them with higher-priority pairs that are formed later. Since men and women are symmetrical, a stable solution can be obtained even if they are swapped. In that case, the difference appears in which side's preferences are prioritized when there are multiple stable solutions. To shorten the notation, we denote the state of not having a pair as unmarried.

Problem **The Stable Marriage Problem** (Southeastern Europe 2007)

Given the preference lists of N men and N women, create a stable matching that prioritizes the men's preferences.

Men are represented by a single uppercase letter, and women are represented by a single lowercase letter. Each preference list contains N people. The output should be processed in the order of the men's names.

http://poj.org/problem?id=3487

Since each man and woman is given by name, it is slightly more complex than when given by numbers such as 0..N. Here, we use the fact that a single alphabet character is equivalent to a numerical value in the range $[0, 127]$ and treat it as a number. The preference list of man 'a' is represented by the first N characters of pref_m['a'], and similarly, the preference list of woman 'A' is represented by pref_m['A']. That is, for man 'a', the most preferred woman is pref_m['a'][0], the next is pref_m['a'][1], and so on. The unused parts of the table are filled with NULL characters (automatically converted from 0 in C++).

C++

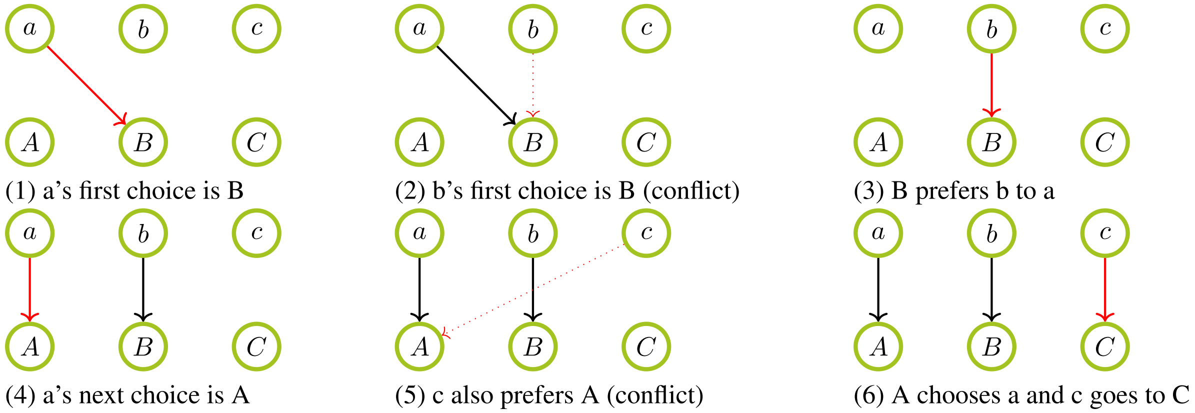
```
#include <algorithm>
// In the first sample, pref_m['a'] is BAC...  (only the first N characters are used, others are null)
```

```
char pref_m[256][256]={{0}}, pref_f[256][256]={{0}};

// Returns true if woman f prefers man a to man b
bool better(char f, char a, char b) {
  int orda = find(pref_f[f], pref_f[f]+27, a) - pref_f[f]; // Rank of a in the list
  int ordb = find(pref_f[f], pref_f[f]+27, b) - pref_f[f]; // Rank of b
  return orda < ordb;
}
```

Next, create the input/output. It is a bit complex, so even if you do not understand every detail of the following sample, it is okay if you can confirm that it works. `scanf(" %c",c)` is a syntax that skips whitespace and reads one character.

C++

```
#include <cstdio>
#include <iostream>
#include <string>
using namespace std;
int T, N;
int main() {
    scanf("%d", &T);
    for (int t=0; t<T; ++t) {
        scanf("%d", &N);
        string name_m, name_f;
        char c;
        for (int i=0; i<N; ++i) {
            scanf("␣%c", &c);
            name_m += c;
        }
        for (int i=0; i<N; ++i) {
            scanf("␣%c", &c);
            name_f += c;
        }
        sort(name_m.begin(), name_m.end());

        for (int i=0; i<N; ++i) {
            scanf("␣%c:", &c);
            scanf("␣%s", pref_m[c]);
        }
        for (int i=0; i<N; ++i) {
            scanf("␣%c:", &c);
            scanf("␣%s", pref_f[c]);
        }
        if (t>0) puts("");
        // Before solving the problem, let's output pref_m and pref_f
        for (int i=0; i<N; ++i)
            cout << name\_m[i] << pref\_m[name\_m[i]] << endl;
        for (int i=0; i<N; ++i)
            cout << name\_f[i] << pref\_f[name\_f[i]] << endl;
        // Then combine it to call the solve() function around here
    }
}
```



(1) a's first choice is B
(2) b's first choice is B (conflict)
(3) B prefers b to a
(4) a's next choice is A
(5) c also prefers A (conflict)
(6) A chooses a and c goes to C

The Stable Marriage Problem: Example of operation in the first sample

Next, implement the algorithm. The current pair of man `'a'` is represented by `m2f['a']` (NULL character if not present). Similarly, the current pair of woman `'A'` is represented by `f2m['A']`.

The number of women that man `'a'` has proposed to in his list is managed by `proposed['a']`.

C++

```
void solve(string name_m, string name_f) {
    char m2f[256]={0}, f2m[256]={0};
    int proposed[256]={0};
    int man_id = 0; // Index of the man
    while (true) {
        char man = name_m[man_id]; // Name of the man
        if (m2f[man]) { // Already assigned
            ++man_id; // Move to the next person
            if (man_id == N) break;
            continue;
        }
        // man does not have a pair yet
        while (!m2f[man]) {
            char f =  pref_m[man][proposed[man]]; // Select the woman of
"next priority" for man
            proposed[man]++; // Increment the "next priority" of man
            char mate = f2m[f]; // Current pair of f
            if (! mate || better(f, man, mate)) {
                if (mate) m2f[mate] = 0; // Cancel the pair of mate and
f
                m2f[man] = f; // Record the pair of man and f in m2f
                f2m[f] = man; // Record the pair of man and f in f2m
                man_id = 0; // Start over from the beginning
            }
        }
    }
    for (int i=0; i<N; ++i)
        printf("%c␣%c\n", name_m[i], m2f[name_m[i]]);
}
```

Chapter 18

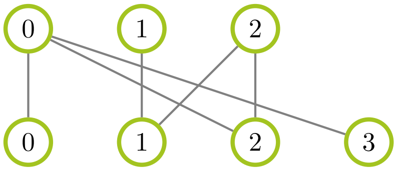
Bipartite Graph Matching

Let's consider the problem of finding a maximum matching in a bipartite graph. This problem is similar to the previous section, but differs in that there are no preference rankings and it is not guaranteed that everyone can be paired. Determining temporary pairs and reassigning them as needed is similar to the previous section, but the decision of whether or not to reassign requires recursion. (Reference[3, pp. 195–])

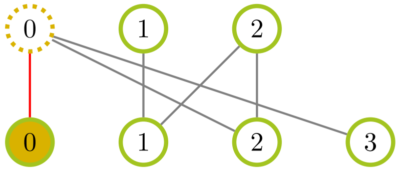
Example **Matching - Bipartite Matching** (AOJ)

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=GRL_7_A&lang=jp

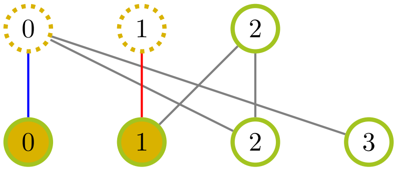
An example of the operation with sample input 1 is shown below.



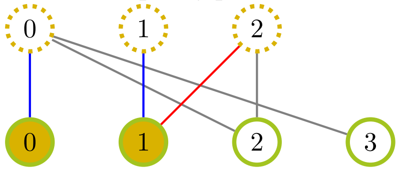
Initial state



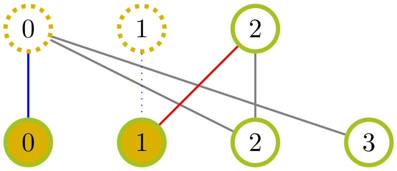
Create a temporary pair from 0 above



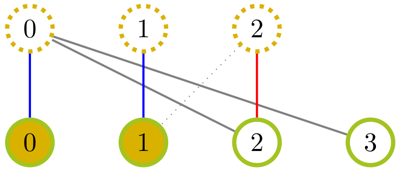
Create a temporary pair from 1 above



A conflict occurs when trying to create a temporary pair from 2 ab



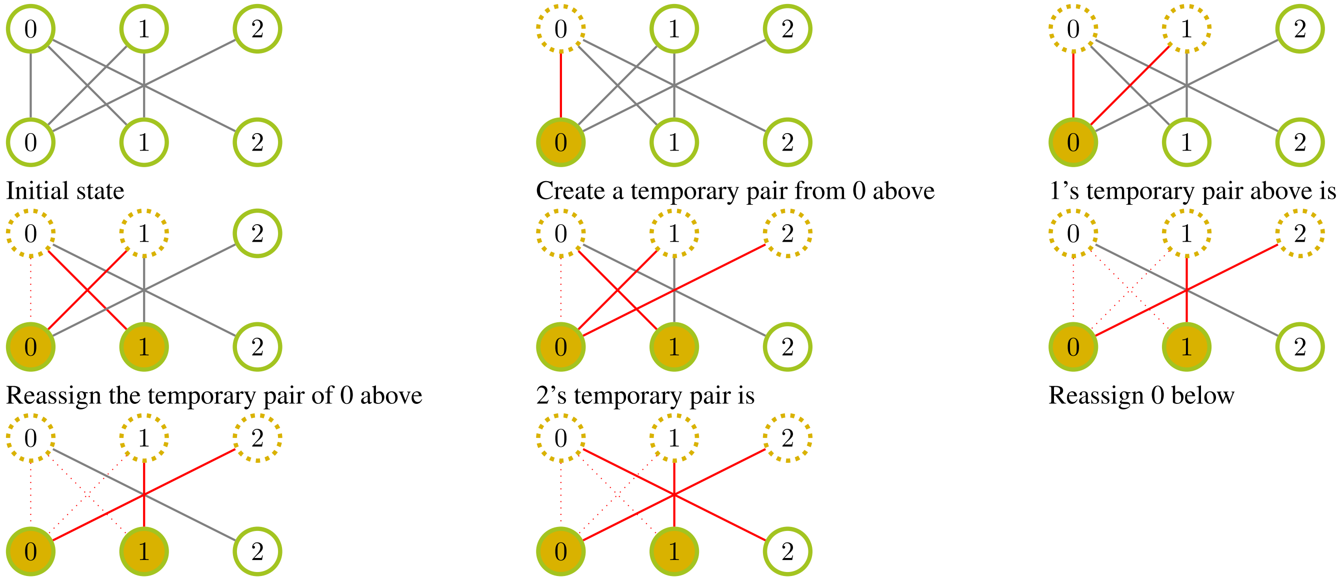
Even if 1’s pair is rearranged, 1 above is isolated



Return 1’s temporary pair and 2 makes a different pair

Let’s also check a slightly more complex example where reassignment occurs.

```
3 3 6
0 0
0 1
0 2
1 0
1 1
2 0
```



Initial state

Create a temporary pair from 0 above

1's temporary pair above is

Reassign the temporary pair of 0 above

2's temporary pair is

Reassign 0 below

Reassign 1 above

Reassign 0 above

Input example

C++

```cpp
int X, Y, E, x, y; // Problem input
bool D[110][110] = {}; // Adjacency matrix
int main() {
  cin >> X >> Y >> E;
  for (int i=0; i<E; ++i) {
    cin >> x >> y;
    D[x][y] = true;
  }
  // Calculate here
}
```

Calculation part

C++

```cpp
int PY[110]; // Current pair of Y
bool V[110];
bool match(int x) {
  if (x<0) return true;
  if (V[x]) return false;
  V[x] = true;
  for (int y=0; y<Y; ++y) {
    if (!D[x][y]) continue;
    if (match(PY[y])) {
      PY[y] = x;
      return true;
    }
  }
  return false;
}
int main() {
  ... // Assume input is finished
  fill(PY, PY+Y, -1); // Initialize current pairs
  int count = 0; // Number of matches
  for (int x=0; x<X; ++x) { // Try sequentially from the X side
    fill(V, V+X, false);
    if (match(x)) ++count; // If at least one assignment starting from x
succeeds
  }
  ... // Output
}
```

Problem **Cards⋆** (National Preliminary 2009)

Pair blue and red cards and remove them. What is the maximum number that can be removed? http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1163&lang=jp

C++

```
int main() {
    while (input) {
       // Set all red card pairs to none (-1)
       P[all red] = -1;
       // C[blue][red]=1 (if pairable), C[blue][red]=0 (if not pairable)
       setting
       for (all blue cards: blue)
           for (all red cards: red)
               C[blue][red] = (__gcd(B[blue], R[red]) >= 2);
       // Match one blue card at a time
       count = 0;
       for (all blue cards: blue) {
           V[all red] = false;
           if (match(blue)) ++count;
       }
       count is the answer;
    }
}
```

C++

```
// V[red] ..  true if card red has no new assignment, false if unknown
// P[red] ..  blue card number paired with card red, -1 if no assignment
bool match(int blue) { // Can blue be paired without reducing the number of pairs so far?
    if (blue < 0) return true;
    for (all red cards: red) {
       if (``blue'' and ``red'' cannot be paired || V[red] is true) continue;
       V[red] = true;
       if (match(P[red])) { // If a different pair can be found for P[red]
         P[red] = blue; // The new pair of ``red'' is blue
         return true;
       }
    }
    return false;
}
```

Problem **Defend the Bases⋆** (Summer Camp 2009)

Move the troops and defend all the bases. (Although not specified, it seems that there are enough troops)

`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2161&lang=jp`

18.1 Various Matchings

Problem **Skiers⋆** (9th Polish Olympiad in Informatics)

Find the minimum number of times you need to climb to the top to inspect all the courses in a ski resort.

(Problem continued)

http://main.edu.pl/en/archive/oi/9/nar

Problem **Dangerous Tower** (I)

want to build a tall tower. http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2293&lang=jp

Problem **Travel Agency**⋆⋆ (Algorithmic Engagements 2006)

People who want to travel together and people who don't want to travel together (!) are modeled numerically. Find the optimal travel members.
http://main.edu.pl/en/archive/pa/2006/biu

Problem **Matching**⋆⋆ (Central European Olympiad in Informatics 2011)

Select a series of buildings where a series of logos can be placed.
http://main.edu.pl/en/archive/ceoi/2011/mat

This problem is closer to string matching.

Problem **Ice Skates** ()

Assignment of ice skates.
http://main.edu.pl/en/archive/oi/16/lyz

Chapter 19

Combinatorial Games

19.1 Impartial Games

Some combinatorial games seem to have relatively frequent examples in problem sets. For some games, such as the solution to Nim, Grundy numbers (Reference[3, pp. 272–]), and surreal numbers (https://www.youtube.com/watch?v=Us8UDUqgiLo), it is possible to efficiently determine the overall winner by analyzing subgames. Subgames correspond to situations where moves in one part do not affect other parts, such as the piles in Nim or the endgame in Go (after the major life-and-death situations have been decided). For more detailed information, see, for example, http://www.math.ucla.edu/~tom/Game_Theory/Contents.html.

Example [(N)

im (Stanford Local 2005)] For a given situation, find the number of winning moves.
http://poj.org/problem?id=2975

Problem [(C)

hristmas Game (POJ Founder Monthly Contest 2008.12.28)]

A two-player game. There is a graph that is roughly a rooted forest: cycles only exist at the ends, and the cycles do not share edges. A single move is ”cut one edge and discard the subgraph on the side that is not connected to the root.” (There may be two-sided polygons, etc.).
http://poj.org/problem?id=3710

Problem [(C)

hef Game (November Long Challenge 2013)] Consider a game where numbers are stacked in piles, and players choose numbers (odd or even) that they are responsible for and remove them and everything after.

Answer whether the first player should choose even or odd numbers, or neither.

http://www.codechef.com/NOV13/problems/CHEFGM/

Problem [(U)

nfair Game (Spring Camp 2014)] Nim where the upper limit of the number that can be removed differs depending on the player.

http://jag2014spring.contest.atcoder.jp/tasks/icpc2014spring_j

19.2 Problems Solved by Exhaustive Search

Example [(A)

aron and Bruce] Given a graph, find how many turns (or infinite) A(aron) and B(ruce) can continue to escape when moving alternately.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2108&lang=jp

Problem [(H)

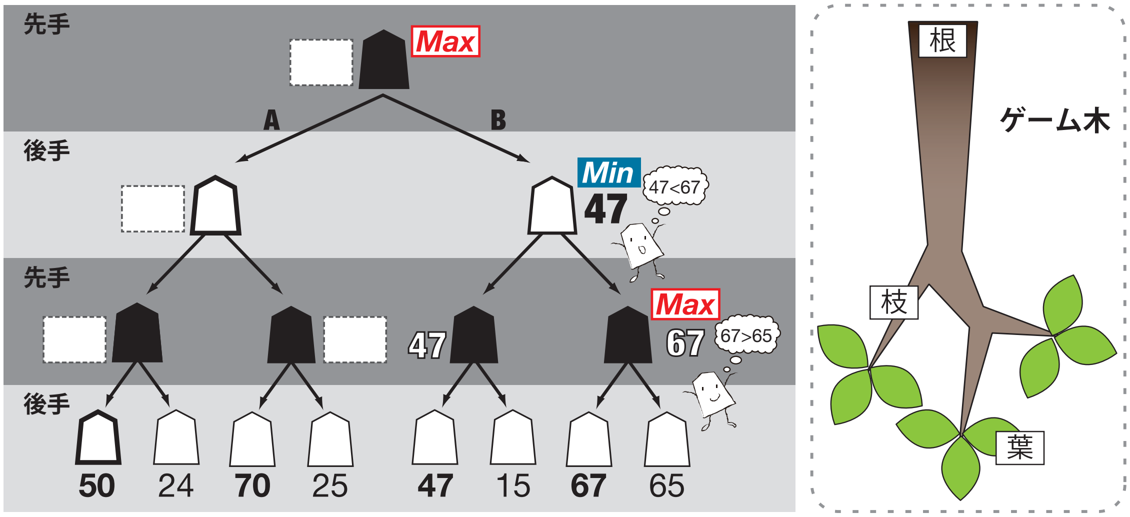
ouse of Cards (World Finals 2009)] Play a game of stacking cards. Who is the winner?

https://icpcarchive.ecs.baylor.edu/index.php?option=com_onlinejudge&Itemid=8&category=43&page=show_problem&problem=2452

Data Supplement

`Axel`

```
13
1R 2R 3R 4R 5R 6R 7R 8R 9R 10R 11R 12R 13R 1B 2B 3B 4B 5B 6B 7B 8B 9B 10B 11B 12B 13B
Case 1: Axel loses 30
```



A と **B** どちらを指すと先手は得？

- Explanation of min-max tree http://en.wikipedia.org/wiki/Minimax
- There are cases where it is appropriate to go back from solved states (Retrograde analysis, c.f., complete analysis of Dobutsu Shogi) and
- cases where it is more appropriate to investigate the future from the state you want to solve (search + $\alpha - \beta$ pruning, etc.). $\alpha - \beta$ pruning is most efficient when reading in order from the best move, and the number of search nodes becomes about the square root of the case where all are read. http://en.wikipedia.org/wiki/Alpha

Various things

- Representation of the state: If the state (situation) is represented in a small way, copying is easy. Breadth-first and best-first searches are also possible. If you use a large representation, prepare one globally and make-move/unmake move. Only depth-first search is suitable. If necessary, do iterative deepening.
- How to create an evaluation function: Easy for games that increase the sum of numbers like the above “House of Cards”. Evaluation functions for Othello and Shogi are created statistically from a large amount of data, so they probably won’t appear in short contests.

- Pay attention to merges and loops: In most games, the game ”tree” is not a tree. When handling merges, a state table (transposition table, in short, a dp table) is required. Loops make the story even more complicated. Especially when the history affects the outcome (a draw on the 4th occurrence of the same state, but the attacker loses in the case of repeated continuous checks, or Ko in Go), if the effect of the history is not considered when writing the outcome in the state table, the outcome may be incorrect (GHI, graph history interaction problem). If possible, Retrograde analysis is safer.

Other:

- Monte Carlo tree search (MCTS, UCT) Probably won’t appear in short contests? `http://en.wikipedia.org/wiki/Monte_Carlo_tree_search`
- Proof-number search: A type of best-first search that expands from where it seems necessary for proof `http://webdocs.cs.ualberta.ca/~mmueller/ps/ICGA2012PNS.pdf`, `http://web.archive.org/web/20041204235835/http://www.cs.vu.nl/~victor/thesis.html`

19.3 Other Problems

Problem [(O)

nonokomachi’s Edit War] Mathematical expression editing game.
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2512&lang=jp`

Problem [(C)

hat noir (UTPC 2009)] Can the cat escape?
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2196&lang=jp`

Problem [(G)

ame Strategy (World Finals 2014)] Alice and Bob’s game.
`https://icpcarchive.ecs.baylor.edu/index.php?option=com_onlinejudge&Itemid=8&category=590&page=show_problem&problem=4785`

Problem [(I)

yasugigappa] A game for three people (?) : Frog, Kappa, and Weasel.
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2557&lang=jp`

Problem [(S)

weet War] A two-player game where players eat or pass chocolates at the beginning of a sequence. Determine the winner.
`http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1353&lang=jp`

Problem [(C)

ircular Game] Players move pieces in turns; the player who cannot move loses. Who wins?
`http://main.edu.pl/en/archive/pa/2009/gra`

Problem [(T)

ermites] A sequence of numbers is given. Players A and B take turns adding a number adjacent to a 0 in the sequence to their score and changing that number to 0. Find the scores of both players when they play optimally.
`http://main.edu.pl/en/archive/pa/2010/ter`

It is probably safer to use scanf.

Part III

Appendix

Appendix A

Bugs and Debugging

It would be ideal if a program worked exactly as intended the first time it was written, but that is often not the case. It takes only a moment to introduce a bug, but it can often take more than two hours to remove it. Furthermore, when using C or C++, the detection mechanisms and error messages for code that is not guaranteed to work are often unhelpful, which can make it take a long time to find the cause. Familiarizing yourself with the useful tools available in the development environment may reduce the time spent investigating the cause. Especially in programming contests, where both time and computer usage are limited, it is desirable for teams to determine efficient methods.

A.1 Bug Prevention and Programming Practices

Paradoxically, to reduce debugging time, it is effective to spend time preventing bugs from being introduced. One way to do this is to adopt good programming styles. There are various books on this topic, so it is good to find one that suits you. Here are some examples.

Variable Usage Using variables appropriately makes programs easier to read.

Bad Example: (Calculating the area of a circle where (x1, y1) and (x2, y2) are the endpoints of the diameter)

C++

```cpp
double area = sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1))/2
 * sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1))/2 * 3.1415;
```

Improved Example:

C++

```cpp
double radius = sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1))/2;
double area = radius*radius*3.1415;
```

Why it's better:

- Easier for humans to read (reduces the places to check if x1 is mistaken for x2)
- No typing/copy-paste errors
- Easier to grasp intermediate results (in this case, the radius). (Easy to display with a debugger or printf)

Function Usage Similarly, it leads to more readable programs.

C++

```
double square(double x) { return x*x; }
double norm(double x1, double y1, double x2, double y2) {
  return square(x2-x1)+ square(y2-y1);
}
double circle_area(double r) { return r*r*3.1415; }
```

Constant Usage Giving names to constants also improves readability.

C++

```
const double pi = 3.1415;
const double pi = atan2(0.0,-1.0);
```

Keep Variable Scope as Short as Possible The shorter the scope of a variable, the better. Relatedly, avoid reusing variables, and it is good to use one variable for only one purpose.

C++

```
int i, j, k;
// Suppose there is a function that uses j or k around here
...
int main() {
  for (i=0; j<5; ++i) // Oh no!
    cout << "Hello,␣world" << endl;
  ...
}
```

Declaring the necessary variables for each function greatly improves the situation.

C++

```
int main() {
  int i;
  for (i=0; j<5; ++i) // Compilation error
    cout << "Hello,␣world" << endl;
}
```

However, in modern languages such as C++ and Java, you can declare variables used in loops within the for statement, so this is recommended.

C++

```
int main() {
  for (int i=0; i<5; ++i)
    cout << "Hello,␣world" << endl;
}
```

Avoid Common Pitfalls There are some things that are good to learn in advance. C Language FAQ `http://www.kouno.jp/home/c_faq/` Especially 16 Strange Problems

Understand Compiler Messages Compilers sometimes give helpful advice. Do not ignore them.

- “if-parenth.cc:8:14: warning: suggest parentheses around assignment used as truth value”

C++

```
  if (a = 1) return 1;
  if (a =! 1) cout << "ok";
```

- “no return statement in function returning non-void [-Wreturn-type]”

C++

```
int add(int a, int b) {
  a+b; // Correctly return a+b;
}
```

A.2 Debugging Tools

A.2.1 Tool: assert

In major languages such as C, C++, Python, and Java, a mechanism called `assert` is provided for understanding programs and testing during execution. The assert statement checks a conditional expression during program execution, and if it is true, does nothing; if it is false, it displays an error message and stops.

Code Example Calculation of factorial:

C++

```cpp
int factorial(int n) {
  if (n == 1)
    return 1;
  return n * factorial(n-1);
}
int main() {
  cout << factorial(3) << endl; // Outputs 3*2*1 = 6
  cout << factorial(-3) << endl; // If you accidentally enter a negative number, it won't stop
}
```

The above function works correctly only when the argument n is positive. We want to ensure that the argument n is positive at runtime. To do this, include the cassert header and add an assert statement. As you can see, the content you want to guarantee is described as a conditional expression within the parentheses of assert.

C++

```cpp
#include <cassert> // Added
int factorial(int n) {
  assert(n > 0); // Added
  if (n == 1)
    return 1;
  return n * factorial(n-1);
}
```

In this way, when writing a program based on some ”premise,” it is good to ”assert” that in the source code for better clarity.

If you call `factorial(-1)` in this way, it will display an error and stop.

```
Assertion failed: (n > 0), function factorial, file factorial.cc, line 3.
```

Since `assert` is a runtime test, it can cause a decrease in execution speed. For this reason, a method is provided to easily disable all asserts without changing the source code. For example, define the NDEBUG macro *before* including the cassert header, as follows:

C++

```cpp
#ifndef NDEBUG
#  define NDEBUG
#endif
#include <cassert>
```

assert can also be used in Python. If you start with the option -O, such as `python3 -O`, the `assert` statement will be disabled.

Python3

```
def factorial(n):
    assert n >= 0
    if n == 0:
        return 1
    return n*factorial(n-1)
```

assert can also be used in Java.

Java

```
public class Main {
    static int factorial(int n) {
        assert n >= 0;
        if (n == 0) return 1;
        return n*factorial(n-1);
    }
    public static void main(String[] args) {
        System.out.println(factorial(3));   // This will succeed
        System.out.println(factorial(-3)); // This will...
    }
}
```

In the case of Java, assert is enabled (only when it is added) by adding `-ea` as a runtime option.

```
$ java -ea Main
6
Exception in thread "main" java.lang.AssertionError
        at Main.factorial(Main.java:3)
        at Main.main(Main.java:10)
```

A.2.2 Tool: -fsanitize

In relatively new C++, compiler options such as `-fsanitize=undefined` and `-fsanitize=address` are available. These options are useful when you have written an ”incorrect program, but you don’t know it, or you can’t identify the location.”

The function `fib` in the following program has a bug where it does not return a value when the argument `i==0`.

C++

```cpp
#include <iostream>
using namespace std;
int fib(int i) {
  if (i == 1) return 1; // Forgot the case for i==0
  else if (i >= 2) return fib(i-1)+fib(i-2);
}
int main() {
  cout << fib(2) << endl;
}
```

When the -fsanitize=undefined option is enabled as shown below, a runtime error is detected.

```
$ g++ -Wall fib.cc
$ ./a.out
2
$ g++ -Wall -fsanitize=undefined fib.cc
$ ./a.out
fib.cc:3:5: runtime error: execution reached the end of a value-returning function withou
Abort trap: 6
```

The following (careless) program works correctly only when the integer read from the keyboard is between 0 and 3 inclusive.

C++

```cpp
1   #include <iostream>
2   using namespace std;
3   int A[4] = {0,1,2,3};
4   int f(int idx) {
5     return A[idx];
6   }
7   int main() {
8     int idx;
9     cin >> idx;
10    cout << f(idx) << endl;
11  }
```

If you compile with -fsanitize=undefined, it will tell you that a violation occurred on line 5.

```
$ g++ -Wall -fsanitize=undefined test.cc
$ ./a.out
3
3
```

```
$ ./a.out
8
test.cc:5:10: runtime error: index 8 out of bounds for type ’int␣[4]’
```

`-fsanitize=undefined` is useful, but it is not a panacea. Memory-related runtime errors can sometimes be found with `-fsanitize=address`. The following program accesses an invalid address when i=4.

C++

```cpp
#include <iostream>
using namespace std;
int A[4]={0,1,2,3};
int main() {
  int *p = A;
  for (int i=0; i<5; ++i) // 5 is a mistake for 4
    cout << *(p+i) << endl;
}
```

Even if you use `-fsanitize=undefined`, this invalid memory access is not detected, and the program runs as if nothing happened.

```
$ g++ -Wall -fsanitize=undefined addr.cc
$ ./a.out
0
1
2
3
8842826
```

If you compile with `-fsanitize=address`, the invalid memory access is detected.

```
$ g++ -Wall -fsanitize=address addr.cc
$ ./a.out
0
1
2
3
=================================================================
==37898==ERROR: AddressSanitizer: global-buffer-overflow on address 0x000107f8d110 at pc
READ of size 4 at 0x000107f8d110 thread T0
    #0 0x107f8c8e9 in main (a.out:x86_64+0x1000018e9)
    #1 0x7fff50332014 in start (libdyld.dylib:x86_64+0x1014)
```

A.2.3 Tool: _GLIBCXX_DEBUG (G++)

In the case of G++, if you define _GLIBCXX_DEBUG at the beginning, it will find some mistakes. (http://gcc.gnu.org/onlinedocs/libstdc++/manual/debug_mode_using.html#debug_mode.using.mode)

C++

```cpp
#define _GLIBCXX_DEBUG
#include <vector>
using namespace std;
int main() {
    vector<int> a;
    a[0] = 3; // Violation of assigning to a vector of length 0
}
```

Execution example: (Instead of simply causing a segmentation fault, it tells you that it is out-of-bounds)

```
/usr/include/c++/4.x/debug/vector:xxx:error: attempt to subscript container
    with out-of-bounds index 0, but container only holds 0 elements.
```

A.2.4 Tool: gdb

Suppose you accidentally write a for loop that doesn't stop, like this:

C++

```cpp
int main() { // Intending to repeat "hello" and "world" with newlines
  for (int i=0; i<10; ++i) {
    for (int j=0; j<2; ++i)
      cout << "hello␣" << endl;
    cout << "world" << endl;
  }
}
```

To prepare to use gdb, add the -g option to the compilation options.

```
$ g++ -g -Wall filename.cc
```

When executing, start gdb by giving it the name of the program to be debugged, and type run inside gdb.

```
$ gdb ./a.out
(gdb starts)
(gdb) run # (Normal execution)
(gdb) run < sample-input.txt # (When using redirection)
# ...(The program executes)...
# ...(Stop by typing Ctrl-C or due to a segmentation fault, etc.)
(gdb) bt
(gdb) up // Go up to main a few times
(gdb) up
#12 0x080486ed in main () at for.cc:6
6             cout << "hello␣" << endl;
(gdb) list
1       #include <iostream>
2       using namespace std;
3       int main() {
4         for (int i=0; i<10; ++i) {
5           for (int j=0; j<2; ++i)
6             cout << "hello␣" << endl;
7           cout << "world" << endl;
8         }
9       }
(gdb) p i
$1 = 18047
(gdb) p j
$2 = 0
```

Main commands:

- Display function call relationships: bt
- Display the value of a variable: p variable name
- Move one level up (to the caller): u
- Display source code: list
- Step execution: n, s
- Continue execution: c
- Exit gdb: q

You can also set breakpoints to interrupt execution when a specific location in the source code is reached, or when a variable's value is changed. See the manual for details.

A.2.5 Tool: valgrind

C++

```
int main() {
    int p; // Forgot initialization
    printf("
}
```

Compile with the -g option, as when using gdb.

```
$ g++ -g -Wall filename.cc
```

When executing, give the executable program to the valgrind command.

```
$ valgrind ./a.out
Conditional jump or move depends on uninitialised value(s)
...
```

A.3 Sampling: Once You've Identified the Cause of a Bug

Once you've identified the cause of a bug, sampling it can be used as an asset to reduce future debugging time. If you move on with "I'm lucky it worked," nothing will remain. Although it's off-topic, even in situations where you get stuck due to overlooking problem constraints or misunderstanding the meaning of the text, it would be helpful to collect patterns of misreading.

Array Boundaries

C++

```
    int array[3];
    printf("
```

Uninitialized Variables

C++

```
    int array[3];
    int main() {
      int a;
      printf("
␣␣␣␣}
```

Function without return

C++

```
int add(int a, int b) {
  a+b; // Correctly return a+b;
}
int main() {
  int a=1,b=2;
  int c=add(a,b); // The value of c is undefined
}
```

Stack Overflow

C++

```
  int main() {
    int a[100000000]; // It’s better to move this to a global variable
  }
```

Invalid Pointer

C++

```
int *p;
*p = 1;

char a[100];
double *b = &a[1];
*b = 1.0;
```

Required capacity for strings: A null terminator ’\0’ is needed at the end

C++

```
    char a[3]="abc"; // Correctly a[4] = "abc" or a[] = "abc"
    printf("
```

C++

```
for (unsigned int i=N-1; i>0; ++i)
  cout << A[i] << endl;
```

C

```
int a, b;
scanf("
```

C++

```
int a, b;
cin >> a, b;
```

Appendix B

Understanding Programming Languages and Environments

B.1 Loop Invariants

Let's consider the correctness of a simple `for` statement like the following:

```python
def name(parameter1, parameter2...):
    variable = initial_value
    for ...
      variable = expression # Rewrite the variable to a new value
    return variable
```

Consider the following example problem and its solution.

Example **Sum of Consecutive Numbers (sum_to_n)** (no judge)

Create a function `sum_to_n(n)` that calculates the sum of integers from 1 to n by naively adding them. Assume $1 < n$. (Verify with n=10000, etc.)

Example Solution:

Python3

```
def sum_to_n(n):
    sum = 0
    for i in range(1, n+1):
        sum = sum + i
    return sum
```

Let's trace how such a program is derived.

If we were to calculate the sum up to a fixed value of 3, it could be written simply.

Python3

```
def sum3():
    return 1+2+3
```

Using auxiliary variables s_i (the sum from 0 to i), we can rewrite it into an equivalent program.

Python3

```
def sum3():
    s0 = 0 # Sum of an empty set
    s1 = s0 + 1 # Sum of 1
    s2 = s1 + 2 # Sum of 1 and 2
    s3 = s2 + 3 # Sum of 1, 2, and 3
    return s3
```

Using assignment, we can also express the auxiliary variables s_i with a single variable `sum`.

Python3

```
def sum3():
    sum = 0 # sum is equivalent to s0
    sum = sum + 1 # The sum on the left is equivalent to s1, the sum on the right is equivalent to s0
    sum = sum + 2
    sum = sum + 3
    return sum # sum is equivalent to s3
```

The `for` loop version of this is the example solution at the beginning.

Python3

```
def sum_to_n(n):
    sum = 0
    for i in range(1, n+1):
        # (a) At this point, the value of sum is the sum from 0 to i-1
        sum = sum + i  # Note that sum += i is the same
        # (b) At this point, the value of sum is the sum from 0 to i
    return sum
```

Please confirm that the conditions regarding the value of sum described in comments (a) and (b) hold true at any point in the loop. Such conditions are called `loop invariants`. Using these conditions regarding the value of sum, we can prove that the function calculates the sum up to n.

B.2 Recursion

B.2.1 Inductive Definitions and Linear Recursion

Consider the situation of ”wanting to calculate something repeatedly.” There are two ways to write this: one is by using iteration or **loops** such as `while` or `for`, and the other is by using `recursion` where a function calls itself. Here, we will delve into the latter.

As an example, let's define a function equivalent to $1 + 2 + \cdots + n$ using a `recurrence relation`. If we write this function as $\mathrm{sum}(n)$, it can be defined inductively as follows:

$$\mathrm{sum}(n) = \begin{cases} 1 & \text{when } n = 1 \\ n + \mathrm{sum}(n-1) & \text{otherwise} \end{cases}$$

Note that sum itself is used within this definition. This can be almost directly translated into Python or C++ definitions:

C++

```cpp
int sum(int n) {
  if (n == 1) return 1;
  else return n + sum(n-1);
}
```

The sum in red is the recursive structure.

Python3

```python
def sum1(n):
    if n == 1:
        return 1
    else:
        return n + sum1(n-1)
```

Understanding Program Correctness It is good to confirm the following two points:

- Confirm the correctness of the base case: when $n = 1$, `sum1(n)` $= 1$
- Assuming correctness up to $n - 1$, confirm the correctness for n: `sum1(n)` $= n +$`sum1(n-1)`

B.2.2 Recursion with Branching

In the recursive calls we have seen so far, a function called itself only once. From now on, we will deal with cases where multiple recursive calls are made.

Example **Fibonacci Numbers** (no judge)

The Fibonacci sequence (Fibonacci numbers) is a sequence of numbers formed by ”the sum of the previous two numbers,” such as

$$0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$$

Inductively define the n-th Fibonacci number $\mathit{fib}(n)$. Also, define the function `fib(n)`.

Python3

```python
def fib(n):
    print("fib",n) # Display the argument when the function is called
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2)+fib(n-1)
```

Note that recursion is not the best way to calculate Fibonacci numbers. More efficient ways to find Fibonacci numbers will be discussed separately.

Tail Recursion Consider the following definition of $\mathrm{sum}'(s, n)$, which is a slight modification of the formula:

$$\mathrm{sum}'(s, n) = \begin{cases} s + 1 & \text{when } n = 1 \\ \mathrm{sum}'(s + n, n - 1) & \text{otherwise} \end{cases}$$

Considering s in the above definition as ”the sum so far,” confirm that $\mathrm{sum}(n) = \mathrm{sum}'(0, n)$.

Python3

```python
def sum2(s,n):
    if n == 1:
        return s+1
    else:
        return sum2(s+n, n-1)
```

A structure where a function calls itself as its return value is called tail recursion. Writing it this way has the advantage that it is easier for the compiler to optimize.

Also, introducing auxiliary variables like s can sometimes improve the clarity of recursion.

maximum recursion depth exceeded

Recursive functions must be implemented to stop somewhere. For example, the following function does not have a defined stopping point.

Python3

```
def inf(a):
    return 1+inf(a)
```

If you call this function as `inf(0)`,

```
>>> inf(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in inf
  File "<stdin>", line 2, in inf
  File "<stdin>", line 2, in inf
  [Previous line repeated 995 more times]
RecursionError: maximum recursion depth exceeded
```

you will see an error message indicating that the process could not continue due to `RecursionError: maximum recursion depth exceeded`.

Recursive Thinking about Operations As an example, consider the operation `print_range(n)` that displays integers from 1 to n. Similar to the previous `sum`, the operation can be defined as follows:

$$\mathit{print\_range}(n) = \begin{cases} \text{Display } 1 & \text{when } n = 1 \\ \mathit{print\_range}(n-1)\ \langle\text{afterward}\rangle\ \text{Display } n & \text{otherwise} \end{cases}$$

Python3

```
def print_range(n):
  if n == 1:
    print(1)
  else:
    print_range(n-1)
    print(n)
```

Example **m-ary Numbers** (no judge)

Given integers m and n ($1 \leq n \leq 8$, $1 < m \leq 10$). Create a function `mdigit(m,n)` that displays all n-digit m-ary numbers in ascending order.

```
>>> mdigit(3,2)
00
01
02
10
11
12
20
21
22
```

For simplicity, consider $m = 2$, i.e., binary numbers. For a positive integer n, we want to construct a procedure that achieves the goal based on a procedure that performs the same process for $n-1$. In other words, assuming someone has created a function that displays all $n-1$ digit binary numbers (*1), can we create a function that displays all n digit m-ary numbers based on that?

An n-digit binary number can be decomposed into the leftmost digit (0 or 1) and the remaining $n-1$ digits. Therefore, we want to do something like: display all $n-1$ digit binary numbers with 0 added to the left, and display all $n-1$ digit binary numbers with 1 added to the left.

Based on the above, by slightly adjusting the function (*1), add the string prefix that you want to be written on the left as an argument, and let `B`(prefix, n) be a function that ”generates all n-digit binary numbers for the argument prefix and n, and displays each with the prefix attached.”

$$\texttt{B}(\text{prefix}, n) = \begin{cases} \text{Display prefix} & (n = 0) \\ \text{Execute } \texttt{B}(\text{prefix} + 0, n-1) \text{ and} & \\ \quad \texttt{B}(\text{prefix} + 1, n-1) & (n > 1) \end{cases} \tag{B.1}$$

prefix	3	2,1	
10	0	00	← Realized through B(100,2)
10	0	01	
10	0	10	
10	0	11	
10	1	00	← Realized through B(101,2)
10	1	01	
10	1	10	
10	1	11	

`Operation of B(10,3): B(100,2) and B(101,2)`

Confirm that `B(0,n)` displays all n-digit binary numbers. To handle not only binary numbers but also m-ary numbers, the part that was branching only with 0 and 1 should be branched from 0 to $m-1$ using a `for` statement or similar.

Although the prefix was explained assuming a string (`str`), it can also be represented with `int`. In that case, for example, the operation of adding 1 to the right of the prefix would be `prefix*10+1`.

Syntax: In Python, to display an integer a with N digits, padding with 0s when the number of digits is insufficient, do the following:

Python3

```
>>> print("{:05d}".format(3))
00003
```

B.2.3 Function and Execution State Management

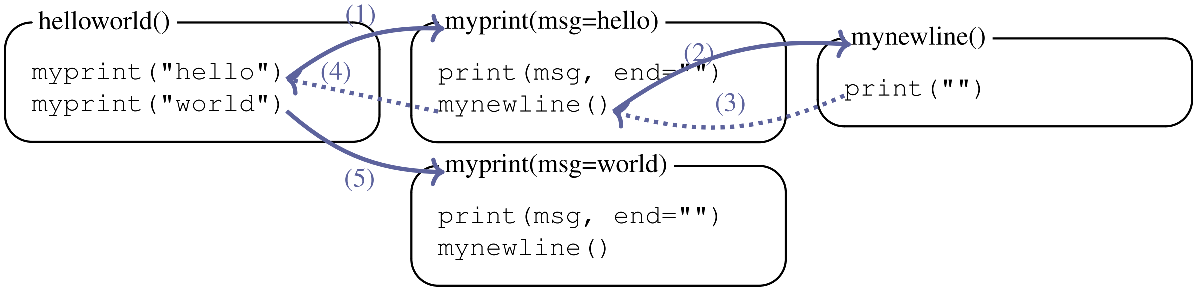
Let's supplement the "relationship between statements being executed from top to bottom and recursive calls."

Normal Function Call Case Although somewhat contrived, let's assume there is the following source code. When the function `helloworld` is executed, `hello` and `world` are displayed line by line.

Python3

```
def mynewline():
    print("")

def myprint(msg): # Writes a message and adds a newline
    print(msg, end="")
    mynewline()

def helloworld():
    myprint("hello")
    myprint("world")
```

A schematic representation of a part of this execution process is as follows. In many programming languages, including C++, when a function is called, a `frame` is created to manage the execution state and local variables of that function. In the figure below, frames are represented by boxes.



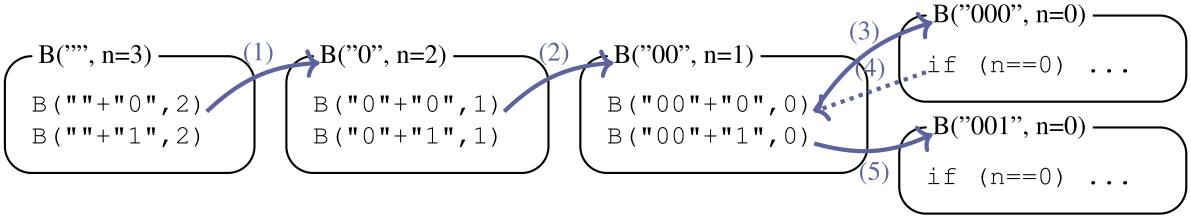
When viewed within each frame, statements are executed in the order they are written. For example, in the leftmost `helloworld` frame, the execution of `myprint("hello")` is completed before the execution of `myprint("world")` begins. The completion of the execution of `myprint("hello")` means that the call to the `myprint` function is executed (arrow (1)), and within that frame, the execution of all statements is completed and control is returned (arrow (4)). In that process, the `mynewline` function is also executed (arrows (2), (3)). As shown by arrows (3) and (4), when the execution of all statements in a frame is completed or a `return` occurs, control is returned to the appropriate location in the caller.

Recursive Case Similarly, in the case of recursion, a frame is created for each call. Suppose the m-ary number example from Section B.2.2 is implemented as follows:

Python3

```python
def B(prefix, n):
    if (n==0):
        ...
        return
    B(prefix+"0", n-1)
    B(prefix+"1", n-1)
```

At this time, the call to `B("",3)` proceeds as follows:



Note that even if it is the same function in the source code, a different frame is created for each call. Each frame maintains information such as arguments, local variables, and how far the execution has progressed. A `stack area` is used to manage such information (different from the stack data structure). Frames that have completed execution are discarded, but frames that have not completed need to be maintained in memory. Therefore, calling a function many times (e.g., 1 million times) may result in a runtime error. For example, in the standard exercise environment, the limit is 8 megabytes.

```
ssh0-01m:~ 0123456789$ ulimit -a
...
stack size              (kbytes, -s) 8192
...
```

B.3 Understanding Integer Types

In this material, we will use the `int` and `long long` types as needed to represent integers. There is a difference in the range of numbers that can be represented by each.

C++

```cpp
long long a = 1000000;
int b = 1000000;
cout << a * a << endl; // 1000000000000
cout << b * b << endl; // -727379968 (overflow)
```

Usually, in C++, integers are represented by variables of type `int`. However, in the iMac environment of this exercise, as shown in Table B.1, the range of values that can be represented by the `int` type and other integer types is limited.

In the previous example,

$$1\,000\,000 \cdot 1\,000\,000 = 10^{12} > 2^{31} \approx 2 \cdot 10^{9}$$

exceeds the range that can be represented by a 32-bit integer. Therefore, in such cases, it is necessary to use the long long type. One of the goals is to be able to grasp this *before starting to write the program*. [1]

The range that can be represented by each type may vary depending on the environment. You can check the environment you are using with the following program. The syntax of `numeric_limits` is omitted, but those who are interested should investigate template classes and standard libraries [6].

C++

[1] It is recommended to memorize $2^{10} = 1\,024 \approx 10^{3}$ in order to perform such calculations smoothly.

Table B.1: Representation of signed integers in the environment assumed by this material

type	bits	Lower Limit	Upper Limit	Notes
int	32	-2^{31}	$2^{31}-1$	Approximately 2 billion
long	32/64			Not recommended for use
long long	64	-2^{63}	$2^{63}-1$	Standard type since C++11, previously a gcc extension
__int128_t	128			gcc extension, not portable but powerful

```
#include <limits>
#include <iostream>
using namespace std;
int main() {
  cout << sizeof(int) // Number of bytes
       <<'␣'<< numeric_limits<int>::digits // Number of bits excluding sign
       <<'␣'<< numeric_limits<int>::digits10 // Number of decimal digits
       <<'␣'<< numeric_limits<int>::min()
       <<'␣'<< numeric_limits<int>::max()
       << endl;
  cout << sizeof(long long)
       <<'␣'<< numeric_limits<long long>::digits
       <<'␣'<< numeric_limits<long long>::digits10
       <<'␣'<< numeric_limits<long long>::min()
       <<'␣'<< numeric_limits<long long>::max()
       << endl;
}
```

```
$ ./a.out
4 31 9 -2147483648 2147483647
8 63 18 -9223372036854775808 9223372036854775807
```

Note that Table B.1 only lists signed integers. Unsigned integers (`unsigned int`, `unsigned long long`, etc.) cannot represent negative numbers, but can represent positive numbers in a range about twice as large. Although they are hardly used in this material, please understand them as needed. In C++11, by using `#include<cstdint>`, types such as int32_t and int64_t can be used. In the future, these should be used, but in this material, we will use `int` and `long long`.

B.4 Floating-Point Numbers and Errors

When dealing with real numbers, floating-point numbers are usually used.

However, depending on the situation, it may be appropriate to use integers to represent decimal numbers. For example, if you only need to handle up to two decimal places (like yen and sen), you can multiply the number by 100 to make it an integer. In that case, you can perform calculations internally as integers and only convert to decimal notation when displaying, such as with `cout << x/100 << "." << (x%100) << endl;`. (Fixed-point)

To flexibly represent larger or smaller numbers, a real number representation (floating-point representation) with a sign part, an exponent part, and a mantissa part is used.

B.4.1 Various Errors

When using floating-point numbers, it is necessary to understand that they involve errors.

Errors are not always intuitive. As an example, consider a loop that processes from 0 to 1 in increments of 0.1. In this case, the appropriate method is to use an integer for the loop counter. Even if they seem equivalent, the method of adding 0.1 each time can lead to unexpected results.

Good Implementation

Python3

```python
i = 0
while i < 10:
  print(i/10.0)
  i += 1
```

```
0.0
0.1
(omitted)
0.8
0.9   # Total of 10 lines
```

Bad Implementation (Displays 11 lines from 0.1...)

Python3

```python
i = 0.0
while i < 1.0:
  print(i)
  i += 0.1
```

```
0.0
0.1
(omitted)
0.7999999999999999
0.8999999999999999
0.9999999999999999      # Total of 11 lines
```

Such phenomena arise from rounding errors, which will be explained next. Furthermore, errors can increase in operations between numbers that already contain errors. There are four types of errors, classified as follows:

Round-off errors Errors caused by the fact that decimal numbers can only be represented approximately with a finite number of digits when expressed in binary (equivalent to the fact that 1/3 cannot be accurately represented with a finite number of digits in decimal notation).
Example: $0.1_{(10)} = 2^{-4} + 2^{-5} + 2^{-8} + 2^{-9} \ldots = 1.10011\ldots_{(2)} \cdot 2^{-4}$
Exercise: Predict the output of the following program?

C++

```
#include <iostream>
using namespace std;
int main() {
  double a = 0.1;
  double b = 0.3;
  if (a*3 == b) cout << "OK" << endl;
  else cout << "NG" << endl;
}
```

```
0.1*3 = 0011111111010011001100110011001100110011001100110011001100110100
  0.3 = 0011111111010011001100110011001100110011001100110011001100110011
```

Loss of significance When taking the difference between two nearly identical numbers represented with a finite number of significant digits, the number of significant digits decreases.
Example: $0.124 - 0.123 = 0.001$
Note: This does not occur in operations between numbers that are originally represented without error.

Information loss In addition or subtraction of numbers with different magnitudes, the smaller number is ignored because it falls outside the significant range of the larger number.
Example: $0.124 + 0.0000000123 = 0.124$

Truncation errors When numerically calculating the value of a function using an infinite series, errors occur due to approximating by truncating at a finite number of terms.

B.4.2 Understanding the `double` Type

In IEEE 754 double-precision (64-bit) floating-point format, a number is represented by a combination of three integers: the sign part s, the exponent part e, and the mantissa part m, as follows:

$$(-1)^s(1 + m \cdot 2^{-52})) \cdot 2^{(e-1023)}. \tag{B.2}$$

The entire representation uses 64 bits, with the sign part s occupying bit 0, the exponent part e occupying bits 1-11 (11 bits), and the mantissa part m occupying bits 12-63 (52 bits).

The following program uses the `union` feature to display the internal representation of a `double` type. Note that `bitset` is used here only to obtain the binary representation of an integer.

C++

```
#include <iostream>
#include <bitset>
using namespace std;
union double_long_long {
  double floating_value;
  unsigned long long integer_value;
};
void show_double(double v) {
  double_long_long u;
  u.floating_value = v;
  cout << v << "␣=␣" << bitset<64>(u.integer_value) << endl;
  long long sign = u.integer_value >> 63;
  long long exponent = (u.integer_value >> 52)& ((1<<11)-1);
  long long mantissa = u.integer_value & ((1ul<<52)-1);
  cout << "␣sign␣=␣" << sign << endl;
  cout << "␣exponent␣=␣" << exponent << '␣' << bitset<11>(exponent) << endl;
  cout << "␣mantissa␣=␣" << bitset<52>(mantissa) << endl;
  cout << "␣=>␣" << (sign ? "-" : "+")
       << "␣(1+" << (double)mantissa/(1ul<<52)
       << ")␣*␣" << "2^(" << (exponent-1023) << ")" << endl;
}
int main() {
  show_double(1.0);
  show_double(-1.0);
  show_double(0.5);
  show_double(0.25);
  show_double(0.75);
}
```

Execution Example: Verify against Equation B.2.

```
1 = 0011111111110000000000000000000000000000000000000000000(omitted)
 sign = 0
 exponent = 1023 01111111111
 mantissa = 00000000000000000000000000000000000000000000(omitted)
 => + (1+0) * 2^(0)
-1 = 101111111111000000000000000000000000000000000000000000(omitted)
 sign = 1
 exponent = 1023 01111111111
 mantissa = 00000000000000000000000000000000000000000000(omitted)
 => - (1+0) * 2^(0)
0.5 = 00111111111000000000000000000000000000000000000000000(omitted)
 sign = 0
 exponent = 1022 01111111110
 mantissa = 000000000000000000000000000000000000000000(omitted)
 => + (1+0) * 2^(-1)
0.25 = 0011111111010000000000000000000000000000000000000(omitted)
 sign = 0
 exponent = 1021 01111111101
 mantissa = 000000000000000000000000000000000000000000(omitted)
 => + (1+0) * 2^(-2)
0.75 = 0011111111101000000000000000000000000000000000000(omitted)
 sign = 0
 exponent = 1022 01111111110
 mantissa = 100000000000000000000000000(omitted)
 => + (1+0.5) * 2^(-1)
```

Exercise: Predict the bit string when 0.875 ($1/2 + 1/4 + 1/8$) is represented as a `double`. Verify using the program above.

B.5 Structures: struct

The `struct` is used to manage data collectively. With the following syntax, a new type called `Student` with integer variables `height` and `weight` is defined:

C++

```
struct Student {
  int height, weight;
};
```

Example of use:

C++

```
  Student a;
  a.height = 150;
  a.weight = 50;
  cout << a.height << '␣' << a.weight << endl;

  Student b = { 170, 70 };
  cout << b.height << '␣' << b.weight << endl;

  Student c = { 180 }; // weight == 0
  Student d = { };  // height, weight == 0
```

- To access the `height` or `weight` of each variable of type Student (in the above, `a, b, c, d`), use the `.` (dot) operator.
- Initialization can also be done with curly braces {}.

Appendix C

Ruby

C.1 Input and Output

This section mainly introduces sample code related to Chapter 3.

C.1.1 ICPC Score Totalizer Software

Ruby

```ruby
while true
  judges = gets.to_i # Input a number
  break if judges == 0
  a = []
  for i in 0..judges-1
    a << gets.to_i
  end # By this point, the scores of each judge are in array a
  sum = 0
  for i in 0..judges-1
    sum = sum + a[i]
  end
  puts sum
  puts a.max
end
```

C.1.2 Array Operations

Ruby

```
  n = 50
  # Prepare an array of length n and initialize each element to 0 (valid from a[0] to a[n-1])
  a = Array.new(n,0)
```

Ruby

```
a = Array.new(5)
a.each{|e| puts e}
```

Ruby

```
a = [3,5,1,2,4]
a.sort!
a.reverse! # Reverse the order of a
p a
```

Ruby

```
a = [0,1,2,3,4,5,6]
a[0,7] = a[3,4]+a[0,3]
p a # ➔  [3, 4, 5, 6, 0, 1, 2]
```

C.1.3 Hanafuda Shuffle

Ruby

```
while line = gets
  n, r = line.split("␣").map{|i| i.to_i} # Read n and r
  break if n == 0
  # Create a deck of n cards
  # Try displaying the created deck
  r.times {
    p, c = gets.split("␣").map{|i| i.to_i}
    # Perform shuffle p, c
    # Try displaying the entire deck after each shuffle
  }
  # Output the top of the deck
end
```

C.2 Sorting and Greedy Algorithms

This section mainly introduces sample code related to Chapter 4.

C.2.1 Sorting

Ruby

```
a = [3,5,1,2,4]
a.sort!
p a
```

C.2.2 Finding Minimum String

Ruby

```
N = gets.to_i
A = []
(1..N).each {
  A << gets.chomp
}
... // Sort A as with integers
puts A[0]
```

C.2.3 Pairs and Sorting

Ruby

```
a = [3,5]
p a # Displays [3,5]
b = [0.5,"X"]
p b # Displays [0.5,"X"]
```

C.2.4 Princess's Marriage

Ruby

```
while line = gets
  pN, pM = line.split("␣").map{|s| s.to_i}
  break if pN == 0
  pd = [] # Array to store <p,d>
  (1..pN).each { # Do something pN times
    d, p = gets.split("␣").map{|s| s.to_i}
    pd << [p, d]
  }
  # Try sorting pd in descending order
  # Try outputting pd
  # If the sorting is successful, try calculating the answer
end
```

C.3 Dynamic Programming

This section mainly introduces sample code related to Chapter 5.

C.3.1 Naive Calculation of Fibonacci Numbers

Ruby

```ruby
def fib(n) # Slow version
  # p ["fib",n] Display the argument when the function is called
  if n==0
    0
  elsif n == 1
    1
  else
    fib(n-2)+fib(n-1)
  end
end
```

C.3.2 Heian-kyo Walking

Input Example

Ruby

```ruby
dataset = gets.to_i
dataset.times {
  gx, gy = gets.split("␣").map(&:to_i)
  p [gx,gy]
  matatabi = gets.to_i
  # In the problem statement, variable p is used, but it is separated from the display command p
  (0..matatabi-1).each{|i|
    x1,y1, x2,y2 = gets.split("␣").map(&:to_i)
    p [x1,y1, x2,y2]
  }
}
```

C.4 Basic Data Structures

This section mainly introduces sample code related to Chapter 7.

C.4.1 Strings and Reading

Ruby

```
word = gets.chomp # Reads a line with gets and removes the newline character at the end with chomp
puts word+word # Displays the concatenated string
```

C.4.2 Stack

In Ruby, we will use `unshift, shift` as `push, pop` respectively.

Ruby

```
stack = [] # (Treating an array as a Stack)
stack.unshift(3) # Add an element to the beginning
stack.unshift(4)
stack.unshift(1)
while stack.size > 0 # While there are elements
  n = stack.shift # Remove from the beginning
  p n
end
```

In Ruby, it is convenient to treat an array as a queue.

Ruby

```
Q = [] # (Treating an array as a Queue)
Q << 3 # Add an element to the end
Q << 4
Q << 1
while Q.size > 0 # While there are elements
  n = Q.shift # Remove from the beginning
  p n
end
# 3, 4, 1 are displayed in this order
```

C.4.3 Priority Queue

In the case of Ruby, it seems that there are currently no libraries available for online judges, so it is necessary to implement a binary heap or similar yourself.

Ruby

```ruby
class PriorityQueue
  def initialize
    # Implement a binary tree with an array.  The left child of the i-th element is 2i+1, and the right child is 2i+2.
    # The parent element is assumed to have a higher priority than either of its children.
    @array = []
    @size = 0
  end
  def push(a)
    # After temporarily placing it at the end, move it up if its priority is higher than its ancestors.
    @array[@size] = a
    heapify_up(@size)
    @size += 1
  end
  def pop()
    # After taking out the root element (highest priority), temporarily place the last element at the root and adjust.
    raise unless @size > 0
    ans = @array[0]
    @size -= 1
    @array[0] = @array[@size]
    heapify_down(0)
    ans
  end
  def size
    @size
  end
  def swap(p,q)
    @array[p],  @array[q] = @array[q], @array[p]
  end
  def equal_or_better(p,q)
    (@array[p] <=> @array[q]) <= 0
  end
  def heapify_up(n)
    while n > 0
      parent = (n-1)/2
      break if equal_or_better(parent, n)
      swap(parent,n)
      n = parent
    end
  end
  def heapify_down(n)
```

```
    while true
      l, r = n*2+1, n*2+2
      break if @size <= l
      child = l
      child = r if r < @size && equal_or_better(r, l)
      break if equal_or_better(n, child)
      swap(child, n)
      n = child
    end
  end
end

Q = PriorityQueue.new
Q.push([50, 1]);
Q.push([20, 2]);
Q.push([30, 3]);
Q.push([10, 4]);
Q.push([80, 5]);

while Q.size() > 0
  cur = Q.pop(); # Extract the smallest element
  p cur
end
```

C.4.4 Strings and Splitting, Joining, Reversing

Ruby

```
word = "hello"
s = word.split("")
(0..s.length).each {|l|
  a = s.first(l)
  b = s.last(s.length-l)
  # p a
  # p b
  puts a.join("")+"␣"+b.join("")
}
```

Ruby

```
a = "hello"
b = a.reverse
puts b
```

C.4.5 Sets

In Ruby, you can achieve similar functionality using `Hash`.[1] For insertion, use `a[element]=1` instead of `a.insert(eleme` for the number or presence of a specified element, use `a[element]` instead of `a.count(element)`; and for the number of elements in the set, use `a.length` instead of `a.size()`.

Ruby

```ruby
all = Hash.new(0)
puts all.length # 0
all["hello"] = 1
all["world"] = 1
all["good␣morning"] = 1
all["world"] = 1
puts all["world"] # 1
puts all["hello!!!"] # 0
puts all.length # 3
```

C.4.6 Associative Arrays

Ruby

```ruby
table = Hash.new(0) # Specified the value to be 0 when the key does not
exist (*)
table["taro"] = 180
table["hanako"] = 160
puts table["taro"] # 180
puts table["ichirou"] # 0 (*)
```

Ruby

```ruby
table["ichirou"] += 1
puts table["ichirou"] # 1
table["ichirou"] += 1
puts table["ichirou"] # 2
```

Ruby

```ruby
phone = Hash.new
phone["taro"] = 123;
phone["jiro"] = 456;
phone["saburo"] = 789;

phone.each{|key,value|
```

[1] Sets are also available, and you can use them by writing `require 'set'`, so those who are interested should investigate.

```
  p [key, value]
}
# ["taro", 123]
# ["jiro", 456]
# ["saburo", 789]
```

In the case of Ruby, a more concise notation than C++ is possible. Specify the variables you want to use in the loop in the `|key,value|` part.

C.5 Graph Traversal

This section mainly introduces sample code related to Chapter **??**.

Reading an Adjacency List (Example Graph)

Ruby

```
N = gets.to_i
G = (1..N).map{ Array.new(N, 0) }
N.times{
  u,k,*v = gets.split(’␣’).map(&:to_i) # u,k are numbers, v is an array
  v.each {|vi| # For each element vi in v
    # Connect (u-1) and (vi-1)
  }
}
G.each{|r|
  puts r.join(’␣’)
}
```

Breadth-First Search (BFS) (Chapter 8.8)

Ruby

```
Q = [0] # Queue (array) with starting point 0
D = Array.new(N,-1)
D[0] = 0 # Distance to the starting point is 0, other distances are -1
while Q.size > 0
  p ["debug", Q] # Check the operation of Q at each step (remove later)
  cur = Q.shift
  (0..N-1).each {|dst|
    if ... then # If it is possible to move from cur to dst, and dst is unvisited
      D[dst] = D[cur]+1
      Q << dst # Push dst onto Q
    end
  }
end
# Display D
```

Depth-First Search (DFS) (Chapter 8.9)

Ruby

```
def dfs(src)
  D[src] = $time # Record the visit time
  $time += 1
  (0..N-1).each {|dst|
    if ... then # If it is possible to move from src to dst, and dst is unvisited
      dfs(dst)
    end
  }
  F[src] = $time # Record the finish time
  $time += 1
  # At the end of the function, return to the parent (blue arrow)
end


D = Array.new(N) # Initial value of D[i] is nil
F = Array.new(N)
$time = 1

(0..N-1).each {|id| # From the node with the smallest number
  if ! D[id] then # If D[id] is unvisited
    dfs(id) # Start dfs
  end
}
# Output
```

C.6 Shortest Paths

This section mainly introduces sample code related to Chapter 9.

Bellman-Ford Algorithm (Section 9.3.2):

Ruby

```
V,E,r = gets.split(’␣’).map(&:to_i) # Input
Edge = []
E.times{
  s,t,d = gets.split(’␣’).map(&:to_i)
  Edge << [s, t, d]
}
Inf = 10000*100000+100 # Maximum if all vertices are traversed
```

```ruby
C = Array.new(V,Inf) # Upper bound of the distance from the starting point to each vertex
C[r] = 0 # Starting point
V.times{ # V times
  count = 0
  Edge.each{|s,t,d| # For all edges (from s to t, cost d)
    if C[s] < Inf && C[t] > C[s]+d
      C[t] = C[s]+d # Update C[t]
      count += 1
    end
  }
  break if count == 0 # If no updates, it's okay to finish
}

V.times{|i|
  puts (C[i] == Inf) ? "INF" : C[i]
}
```

C.7 Numerical Integration

This section mainly introduces sample code related to Chapter 15.

C.7.1 Find the Outlier

Ruby

```ruby
# Assume values are in array $V
def interpolate(n, e)
  sum = 0.0
  $V.each_index{|k|
    next if k == n || k == e # if k is n or E
    p = $V[k]
    $V.each_index{|i|
      p *= 1.0*(n-i)/(k-i) if i != k && i != n && i != e # if i is not k, n, or E
    }
    sum += p
  }
  sum
end
```

Ruby

```
require 'scanf'
# Function definition
while true
  d = (scanf '\%d')[0]
  break if d == 0
  $V = scanf('\%f'*(d+3))
  (0..d+2).each {|i|
    if interpolate(i, -1) == $V[i] # if ignoring i makes everything consistent
      puts i
      break
    end
  }
end
```

C.7.2 Intersection of Two Prisms

Ruby

```
def width(polygon, x) # polygon contains the coordinates of the x-y or x-z plane in order
  w = []
  polygon.each_index{|i|
    p, q = polygon[i], polygon[(i+1) % polygon.length]
    # Consider the edge pq
    if p[0] == x
      w << p[1]
    elsif (p[0] < x && x < q[0]) || (p[0] > x && x > q[0])
      x0, y0 = p[0], p[1]
      x1, y1 = q[0], q[1]
      x0, y0, x1, y1 = x1, y1, x0, y0 if p[0] > x
      w << y0 + 1.0*(y1-y0)*(x-x0)/(x1-x0)
    end
  }
  raise if w.length == 0
  w.max - w.min
end
```

Ruby

```
def volume
  $X.sort!
  $X.uniq!
  sum = 0.0
  xmin = [($P1.min)[0], ($P2.min)[0]].max
  xmax = [($P1.max)[0], ($P2.max)[0]].min
  (0..$X.length-2).each{ |i|
    a, b = $X[i], $X[i+1]
    next unless xmin <= a && a <= xmax && xmin <= b && b <= xmax
    m = (a+b)/2.0
    va = width($P1, a)*width($P2, a)
    vb = width($P1, b)*width($P2, b)
    vm = width($P1, m)*width($P2, m)
    area = (b-a)*(va+4*vm+vb)/6.0 # Integral of the quadratic equation passing through (a,va), (m,vm), (b,vb) in the interval [a,b]
    sum += area
  }
  sum
end
```

Bibliography

[1] Yutaka Watanabe, ”Introduction to C/C++ Programming for Online Judges,” Mynavi, 2014, https://book.mynavi.jp/ec/products/detail/id=25382

[2] Yutaka Watanabe, ”Algorithms and Data Structures for Programming Contest Strategy,” Mynavi, 2015, https://book.mynavi.jp/ec/products/detail/id=35408

[3] Takuya Akiba, Yoichi Iwata, Yoshito Kitagawa, ”Programming Contest Challenge Book” Second Edition, Mynavi, 2012, https://book.mynavi.jp/ec/products/detail/id=22672

[4] Jon Kleinberg and Eva Tardos, ”Algorithm Design”, (Translated by Takao Asano, Yasuhito Asano, Takao Ono, Tomio Hirata), Kyoritsu Shuppan, 2008, http://www.kyoritsu-pub.co.jp/bookdetail/9784320122178

[5] T. Cormen, C. Leiserson, R. Rivest, C. Stein, ”Introduction to Algorithms” 3rd Edition, Comprehensive Edition, (Translated by Tetsuo Asano, Kazuo Iwano, Hiroshi Umeo, Masashi Yamashita, Koichi Wada), Kindai Kagaku Sha, 2013, http://www.kindaikagaku.co.jp/information/kd0408.htm

[6] Bjarne Stroustrup, “The C++ Programming Language” (4th Edition), Addison-Wesley Professional, 2013