CS.20002 문제해결기법

0주차 – 오리엔테이션, 문제 해결의 기초 도구

TA 오태인

2026년 9월 4일

For International Students

- As stated in the course information on the academic system, this course will be conducted in **Korean**.
- You may use electronic devices during class. All lecture materials will be uploaded, and you may use tools such as LLMs to translate assignments. Please feel free to use any resources that help you follow the course.
- Internet access, except to the exam site, is prohibited during the final exam. Therefore, the problem statements will be provided in English.
- If you have any other suggestions for making the course more accessible, please feel free to let us know.

Problem Solving

- 이 과목은 **Problem Solving (PS)**을 다룹니다.
- 이론이나 프로젝트를 중심으로 하는 대부분의 전산학부 과목과 달리,
  - 주어진 문제를 해결하는 기법을 배우고,
  - 이를 활용해 직접 문제를 해결하는 경험을 하게 됩니다.

Problem Solving

- PS 가운데, 정해진 시간 안에 여러 문제를 해결하여 얻은 점수로 순위를 겨루는 대회 형식을 **competitive programming**이라고 합니다.
- 대표적인 대회로 ICPC와 SCPC 등이 있으며, 대학이나 동아리에서 자체적으로 개최하는 대회도 있습니다.

Problem Solving

- 온라인 플랫폼에서도 정기적으로 대회가 열립니다.
- 참가자는 대회 성적에 따라 변하는 레이팅을 통해 실력을 겨룰 수 있습니다.
  - `https://doj.kr/`
  - `https://codeforces.com/`
  - `https://atcoder.jp/`

강의 정보

- 강의자: TA 오태인
- 강의 시간: 금요일 13:00–16:00
- 강의 장소: E3-1 1501호

평가 – 과제

평가는 **S/U 방식**으로 진행되며, 평가 기준 중 과제에 관한 사항은 다음과 같습니다.

- 총 7개의 과제가 있습니다. 각 과제에는 "권장 점수"가 있으며, 모든 과제의 권장 점수를 합한 값이 과제의 S 커트라인이 됩니다.
- 권장 점수는 수업에서 직접 다룬 문제들의 점수 합으로 정할 예정입니다. 특정 과제에서 권장 점수를 받지 못하더라도 모든 과제의 총점으로 평가하므로, 다른 과제에서 권장 점수를 초과해 받은 점수로 만회할 수 있습니다.

평가 – 과제

- i번째 수업에서 출제된 과제는 $i+1$번째 수업에서 풀이하며, $i+2$번째 수업 전날 23:59:00까지 제출할 수 있습니다.
- i번째 수업에서 출제된 과제와 관련하여 어떠한 종류든 부정행위가 의심되는 학생과는 $i+2$번째 수업이 끝난 뒤 면담을 진행합니다. 면담 대상자는 해당 수업이 끝난 직후 호명하며, 면담에 두 차례 불참하면 U를 받습니다.
- 면담 대상자가 되었다는 사실만으로 불이익을 받지는 않습니다. 다만 면담에서 자신이 제출한 코드의 의도나 동작을 **심각하게** 이해하지 못하는 것으로 확인되면 U를 받습니다.

평가 – 기말고사

- 기말고사는 12/11 또는 12/18에 진행할 예정입니다. 과제와 같은 형태의 프로그래밍 문제 해결 시험으로, 시험 시간은 5시간입니다.
- 시험 사이트를 제외한 인터넷 사용은 금지됩니다. 외부 IDE 사용 역시 금지될 가능성이 높으며, 시험 관련 확정 사항은 추후 공지합니다.
- English problem statements will be provided for the final exam.
- 기말고사의 S 커트라인은 과제의 S 커트라인을 본인의 힘으로 넘긴 학생이 받을 것으로 예상되는 점수의 약 $1/2$로 계획하고 있습니다.

평가 – 출석 인정

- 경조사나 증빙 서류가 있는 건강 문제 또는 중요 행사 참여 등 통상적으로 인정되는 결석 사유에 관한 자료를 **기말고사 전날 18:00** 까지 제출하면, 과제 면담 불참을 결석으로 처리하지 않음을 보장합니다.
- 그 밖의 사유로 인한 결석이나 기한이 지난 증빙 자료 제출도 최대한 너그럽게 인정할 예정입니다.
- ICPC 해외 리저널 등 주요 PS 대회 참가로 인한 결석도 과제 면담 불참으로 처리하지 않으며, **기말고사** 역시 출석 인정 및 S 커트라인 충족 처리가 가능합니다. 이 경우 실제 대회 참가 여부 등을 확인하기 위한 여러 서류를 요구할 수 있습니다.

강의 계획

날짜	내용
9/4	오리엔테이션 + 문제 해결의 기초 도구
9/11	동적 계획법
9/18	휴강(KAIST-POSTECH 학생대제전)
9/25	휴강(추석 연휴)
10/2	동적 계획법
10/9	휴강(한글날)
10/16	동적 계획법
10/23	휴강(중간고사)
10/30	자료구조
11/6	자료구조
11/13	휴강(ICPC Seoul Regional)
11/20	고급 문제 해결 기법
11/27	인터랙티브, 투 스텝
12/4	복습 및 기말고사 안내
12/11	기말고사 후보 1
12/18	기말고사 후보 2

이 수업에서 다루지 않는 것

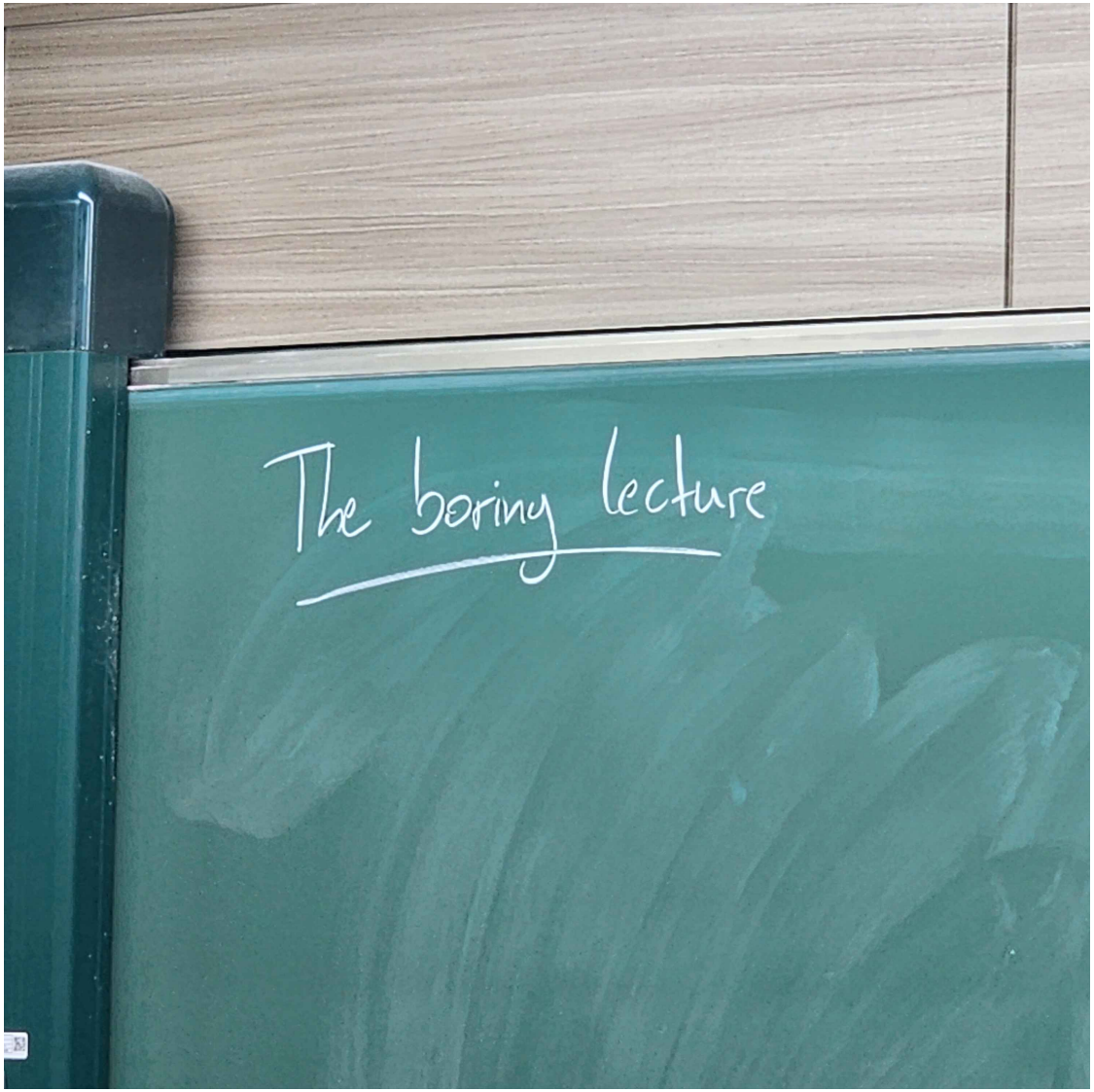
- C++ · Python 등의 기초 문법이나 언어의 내부 구조, 시스템 이론
- 프로그래밍기초, 미적분학 등 기초 과목에서 다루는 내용
  - 관련 개념이 등장하면 짧게 recall합니다.
- 권장선수과목인 이산구조와 데이타구조에서 다루는 내용
  - 관련 개념이 등장하면 짧게 recall합니다.
- 대부분의 그래프 알고리즘
  - 최단 경로, SCC 등은 대부분 알고리즘개론에서 다루는 내용입니다.
  - 더 심화된 그래프 알고리즘을 공부하고 싶다면 Eunjung Kim 교수님과 Sebastian Wiederrecht 교수님께서 개설하시는 과목을 눈여겨보시기 바랍니다.

Break Time

- 12/11 또는 12/18에 다른 수업 혹은 확정된 시험 일정이 있다면 지금 손을 들고 말씀해주세요.
- 곧 KLMS에 설문을 업로드할 예정이니 가능한 시간에 **모두** 투표해주세요.
- 설문 후 확정된 일정은 바꿀 예정이 없으며, 인정되지 않는 이유로 기말고사에 불참하면 0점으로 처리되어 U를 받습니다. 꼭 일정을 확인하시기 바랍니다.

Break Time – English

- If you have another class or a confirmed exam scheduled on either December 11 or 18, please raise your hand and let us know now.
- A poll will be posted on KLMS shortly. Please vote for **all** time slots when you are available.
- Once the exam schedule has been finalized based on the poll, it will not be changed. If you miss the final exam for a reason that is not accepted, you will receive a score of 0 and therefore a U. Please make sure to check your schedule.


The boring lecture

알고리즘 문제 해결이란?

- 주어진 입력에 특정 과정을 수행하여 출력을 내는 프로그램을 작성한다.
- 입력에는 제한이 있으며, 시간과 메모리 사용량에도 제한이 주어진다.

알고리즘 문제 해결이란?

문제 1

n이 주어진다. $1 + 2 + \cdots + n$의 값을 출력하여라.

- 덧셈을 $n - 1$번 수행하면 해결할 수 있다.
- $1 + 2 + \cdots + n = \frac{n(n+1)}{2}$를 사용하면 덧셈, 곱셈, 나눗셈을 각각 1번씩 수행하여 해결할 수 있다.
- 현대의 컴퓨터(우리의 채점기)는 1초에 약 수억 번의 연산을 수행한다.
- 따라서 시간 제한이 1초이고 $n \leq 10^{12}$라면 두 번째 알고리즘만 시간 제한 안에 동작한다.

알고리즘 문제 해결이란?

문제 2

n이 주어진다. n이 홀수라면 $1 + 2 + \cdots + n$의 값을, 짝수라면 $1^2 + 2^2 + \cdots + n^2$의 값을 출력하여라.

- 문제 1과 마찬가지로, 느린 방법과 빠른 방법이 존재한다.
- 그러나 두 방법의 연산 횟수를 각각 정확히 표현하기는 쉽지 않다.
- "대략 n번", "대략 상수 번"을 체계적으로 나타낼 수 없을까?

알고리즘 문제 해결이란?

Big O notation

함수 $f, g\colon \mathbb{N} \to \mathbb{N}$에 대하여, 다음 조건을 만족하면 $f(n)$은 $\mathcal{O}(g(n))$이라고 한다.

$$\exists c \in \mathbb{R}^+,\ \exists N \in \mathbb{N}:\quad \forall n \in \mathbb{N},\ n \geq N \Rightarrow f(n) \leq cg(n).$$

- 계수를 모두 떼고 가장 큰 항을 남긴다고 생각하면 된다.
- $1,\ 2,\ 5,\ 100,\ 998244353 \to \mathcal{O}(1)$
- $n,\ 2n,\ 1+2n,\ \log n + n,\ 2\sqrt{n} + 4n \to \mathcal{O}(n)$
- $n^2,\ 6n + 7n^2,\ 15 + 57n^2,\ n^{\sqrt{2}} + n^2 \to \mathcal{O}(n^2)$

알고리즘 문제 해결이란?

- 프로그램의 동작 시간을 연산 횟수로 나타낸 것을 **시간복잡도**라고 한다.
- 문제 해결에서 자주 등장하는 시간복잡도에는 $\mathcal{O}(A)$, $\mathcal{O}(A^2)$, $\mathcal{O}(2^A)$, $\mathcal{O}(AB)$, $\mathcal{O}(A \log B)$, $\mathcal{O}(A\sqrt{B})$ 등이 있다.
- 입력 제한을 바탕으로 허용되는 시간복잡도를 판단한 뒤 문제를 해결해야 한다.
- 프로그램이 사용하는 공간을 나타내는 **공간복잡도**도 있다. 시간복잡도보다 제약이 되는 경우는 적지만, 메모리 제한 역시 확인해야 한다.

Standard Library

- PS에서는 보통 C++을 사용한다. 널리 쓰이는 언어 중 문법이 비교적 간결하면서 실행 속도가 빠르기 때문이다.
- C 대신 C++을 사용하는 대표적인 이유 중 하나는 **standard library**이다.
- Standard library에는 자주 사용하는 자료구조와 알고리즘이 미리 구현되어 있다.
- PS에서 주로 사용하는 부분은 그중에서도 **Standard Template Library (STL)**이다.
- STL을 제외하면 `iostream` 등 일부만 사용하기 때문에, PS에서는 둘을 엄밀히 구분하지 않는 경우가 많다.

std::stack

- Stack은 원소를 넣고 뺄 수 있으며, 나중에 들어온 원소가 먼저 나오는 자료구조이다.
- STL에는 이러한 자료구조를 구현한 `std::stack`이 있다.
- 주요 메소드로는 다음과 같은 것들이 있다.
  - `push(x)`: 맨 위에 `x`를 넣는다.
  - `pop()`: 맨 위 원소를 제거한다.
  - `top()`: 맨 위 원소를 확인한다.
  - `empty()`: 비어 있는지 확인한다.
  - `size()`: 원소의 개수를 구한다.

std::queue

- Queue는 원소를 넣고 뺄 수 있으며, 먼저 들어온 원소가 먼저 나오는 자료구조이다.
- STL에는 이러한 자료구조를 구현한 `std::queue`가 있다.
- 주요 메소드로는 다음과 같은 것들이 있다.
  - `push(x)`: 맨 뒤에 `x`를 넣는다.
  - `pop()`: 맨 앞 원소를 제거한다.
  - `front()`: 맨 앞 원소를 확인한다.
  - `empty()`: 비어 있는지 확인한다.
  - `size()`: 원소의 개수를 구한다.

std::set

- Set은 중복되지 않는 원소들을 저장하는 자료구조이다.
- STL에는 이러한 자료구조를 구현한 `std::set`이 있다.
- 주요 메소드로는 다음과 같은 것들이 있다.
  - `insert(x)`: 원소 `x`를 넣는다.
  - `erase(x)`: 원소 `x`를 제거한다.
  - `count(x)`: 원소 `x`가 들어 있는지 확인한다.
  - `empty()`: 비어 있는지 확인한다.
  - `size()`: 원소의 개수를 구한다.
- 중복된 원소를 하나로 합치지 않고 모두 저장하는 `std::multiset`도 있다. `set`의 `count(x)`는 0 또는 1이지만, `multiset`에서는 1보다 클 수 있다.

std::set

- set의 내부 구조에는 일반적으로 red-black tree를 사용한다.
- 따라서 set의 크기가 n일 때 원소의 삽입과 삭제에는 $\mathcal{O}(\log n)$의 시간이 걸리며, 원소들은 정렬 정보를 갖고 있다.
- 이를 활용하는 lower_bound와 upper_bound 메소드도 존재한다.
  - lower_bound(x): x 이상인 가장 작은 원소를 가리키는 iterator를 반환한다.
  - upper_bound(x): x보다 큰 가장 작은 원소를 가리키는 iterator를 반환한다.

std::map

- Map은 key와 value의 쌍을 저장하고, key를 이용해 대응하는 value를 찾는 자료구조이다.
- STL에는 이러한 자료구조를 구현한 `std::map`이 있다. 하나의 key에는 하나의 value가 대응한다.
- 주요 사용법은 다음과 같다.
  - `m[key]`: `key`에 대응하는 value에 접근한다.
  - `insert({key, value})`: key와 value의 쌍을 넣는다.
  - `erase(key)`: `key`에 대응하는 원소를 제거한다.
  - `count(key)`: `key`가 들어 있는지 확인한다.
  - `empty()`: 비어 있는지 확인한다.
  - `size()`: 원소의 개수를 구한다.

std::map

- 존재하지 않는 key에 `m[key]`로 접근하면, value의 자료형에 해당하는 기본값으로 초기화된 원소가 새로 생긴다.
- 예를 들어 `map<int, int>`형의 빈 map `m`에 `m[4] += 5;`를 실행하면, `m[4]`가 0으로 초기화된 뒤 5가 더해져 값이 5가 된다.
- `map` 역시 `set`과 마찬가지로 일반적으로 red-black tree를 사용해 구현된다.
- 따라서 `map`의 크기가 n일 때 원소의 삽입, 삭제, 검색에는 각각 $\mathcal{O}(\log n)$의 시간이 걸리며, key들은 정렬 정보를 갖고 있다.
- `lower_bound`와 `upper_bound` 메소드도 key를 기준으로 동작한다.

std::vector

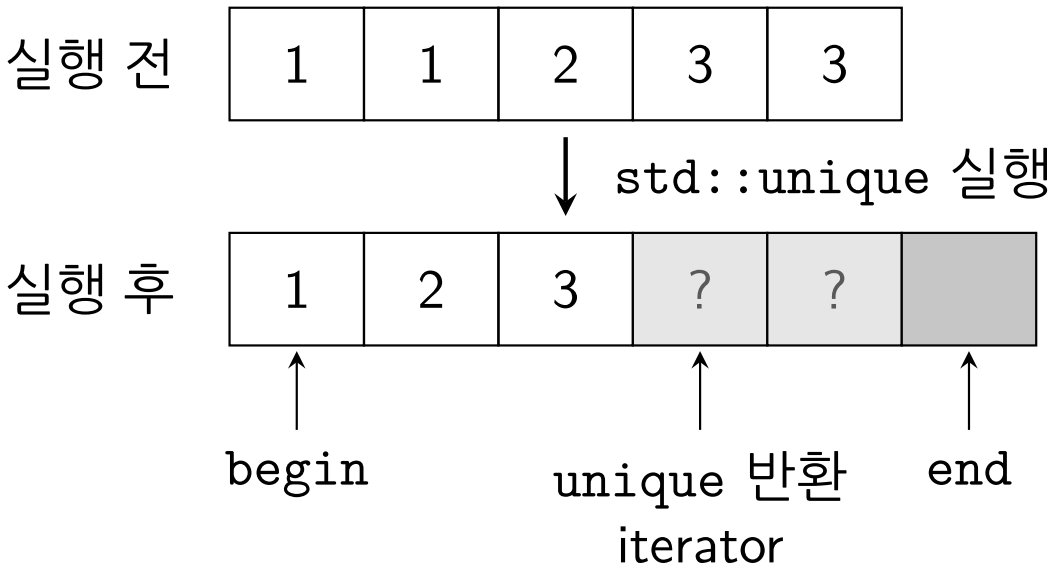
- C에도 `int arr[100];`과 같이 사용하는 배열이 있지만, STL에는 가변 길이 배열을 구현한 `std::vector`가 있다.
- `vector`의 주요 사용법은 다음과 같다.
  - `push_back(x)`, `pop_back()`: 맨 뒤에 원소를 넣거나 제거한다.
  - `back()`: 맨 뒤 원소를 확인한다.
  - `empty()`, `size()`: 비어 있는지 확인하거나 원소의 개수를 구한다.
  - `begin()`, `end()`: 첫 원소와 마지막 원소 다음 위치를 가리키는 iterator를 반환한다.
  - `erase(it)`: iterator `it`이 가리키는 원소를 제거한다.

Iterator

- Iterator란 STL 컨테이너의 원소를 가리키고 순회하기 위한 도구이다.
- C를 다룰 줄 안다면 포인터와 유사하다고 생각하면 된다.
- STL 알고리즘은 보통 $[\texttt{begin}, \texttt{end})$ 형태의 iterator 범위를 사용한다. 즉, `end`는 마지막 원소의 다음 위치를 가리키며 그 위치에는 값이 들어 있지 않다.
- `for (auto it = v.begin(); it != v.end(); ++it)`
  - 첫 원소부터 마지막 원소까지 순회한다. `*it`으로 현재 원소에 접근할 수 있다.

Iterator

- `sort(v.begin(), v.end())`: v의 모든 원소를 오름차순으로 정렬한다.
- `reverse(v.begin(), v.end())`: v의 모든 원소의 순서를 뒤집는다.
- `unique(v.begin(), v.end())`: 연속한 중복 원소를 제외한 결과를 앞쪽에 모으고, 그 결과의 끝을 가리키는 iterator를 반환한다.



std::sort

- sort 함수에는 비교 함수 cmp를 전달할 수 있다.
- sort(vec.begin(), vec.end(), cmp)는 cmp(a, b)가 true이면 a가 b보다 앞에 오도록 정렬한다.
- sort가 올바르게 동작하려면 모든 a에 대해 cmp(a, a) == false 여야 한다.
- 예를 들어, 원소가 음이 아닌 정수일 때 모든 짝수를 홀수보다 앞에 두고, 각각의 그룹 안에서는 오름차순으로 정렬하려면 다음 비교 함수를 사용할 수 있다.

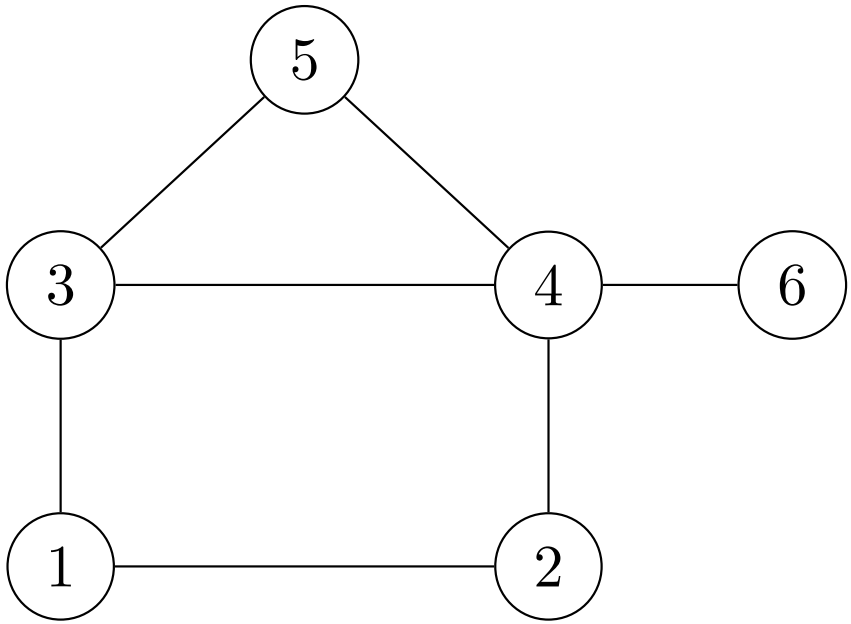
```
bool cmp(int a, int b) {
  if (a % 2 == b % 2) return a < b;
  return a % 2 < b % 2;
}
```

기타 STL 자료형

- STL에는 몇 가지 type이 더 있다. PS에서는 주로 다음과 같은 것들을 사용한다.
- `tuple`: 서로 다른 type의 여러 값을 순서대로 묶을 수 있다. 예를 들어 `tuple<int, int, double, vector<int>>`가 가능하다.
- `array`: 한 type의 고정 길이 배열이다. 예를 들어 `array<int, 6>`은 길이가 6인 정수 배열이다.
- `pair`: 크기가 2인 `tuple`이라고 볼 수 있다. `pair<int, double>` 등이 가능하며, `first`와 `second`로 각 원소에 접근할 수 있다는 차이가 있다.
- 위 자료형들은 모두 사전순으로 대소를 비교한다. 앞 원소부터 비교하면서 값이 같으면 다음 원소로 넘어가고, 다르면 해당 원소의 대소가 전체의 대소가 된다.

STL 사용 예시

- 그래프란 정점과 정점들을 잇는 간선으로 이루어진 이산수학의 대표적인 구조이다.



STL 사용 예시

문제 3

정점이 최대 $N \leq 100{,}000$개, 간선이 최대 $M \leq 1{,}000{,}000$개인 그래프가 주어진다. 가장 많은 정점과 인접한 정점을 하나 찾고, 그 정점과 인접한 모든 정점을 출력하여라.

- 정적 배열을 사용하면 한 정점마다 최대 $N-1$개의 인접 정점을 저장해야 하므로, $\mathcal{O}(N^2)$의 공간을 미리 할당해야 한다.
- `int adj[100000][100000]`은 약 $40{,}000\,\mathrm{MB}$를 사용하므로 통상적인 메모리 제한을 크게 넘는다.

STL 사용 예시

- 대신 다음과 같이 정점마다 하나의 `vector`를 둘 수 있다.

```
vector<int> adj[100005];
```

- 선언 직후에는 정점의 수에 비례하는 $\mathcal{O}(N)$의 공간만 사용한다.
- 간선 uv가 입력되면 `adj[u]`와 `adj[v]`에 각각 상대 정점을 하나씩 추가한다.
- 모든 간선을 입력한 뒤 저장된 원소는 총 $2M$개이므로, 전체 공간복잡도는 $\mathcal{O}(N+M)$이다.

STL 사용 예시

```cpp
#include <bits/stdc++.h>
using namespace std;

int n, m, u, v;
vector<int> adj[100005];

int main() {
  scanf("%d %d", &n, &m);
  while (m--) {
    scanf("%d %d", &u, &v);
    adj[u].push_back(v);
    adj[v].push_back(u);
  }
  int mx = 0;
  for (int i = 1; i <= n; i++) mx = max(mx, (int)adj[i].size());
  for (int i = 1; i <= n; i++) {
    if (adj[i].size() == mx) {
      for (auto j : adj[i]) printf("%d ", j);
      return 0;
    }
  }
}
```

재귀함수

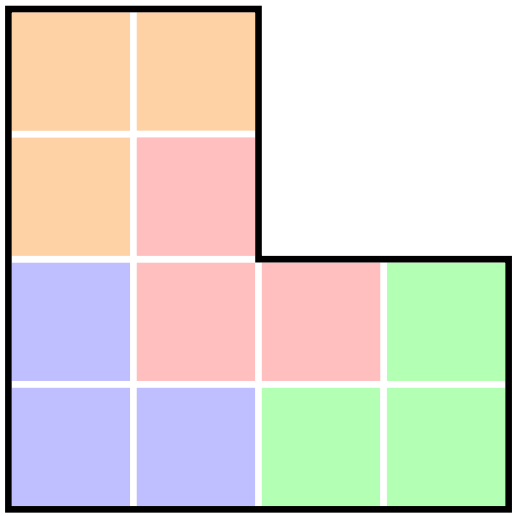
- PS에서 자주 사용하는 문법 중 하나로 재귀함수가 있다.
- 재귀함수는 자기 자신을 호출하는 함수이다. 같은 동작을 여러 번 반복하거나, 더 작은 상태의 문제를 풀어 더 큰 상태의 문제를 해결하는 등의 상황에서 폭넓게 사용된다.
- 프로그램이 종료되려면 재귀함수를 호출하지 않고 끝나는 분기를 올바르게 작성해야 한다.

재귀함수

문제 4

두께가 2^n 인 L자 모양을 두께가 1인 L자 타일로 빈틈없이 채워 보자.

- 두께가 2인 L자 모양은 다음과 같이 네 개의 두께 1인 L자 타일로 완전히 채울 수 있다.

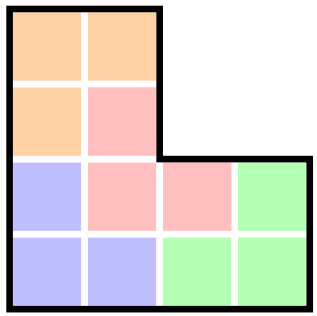


재귀함수

- 같은 방법으로, 두께가 2^n 인 L자 모양을 두께가 2^{n-1} 인 L자 모양 네 개로 나눌 수 있다.

채우기(L, n)

1. $n = 0$이면 L 자체를 하나의 타일로 덮고 종료한다.
2. L을 두께가 2^{n-1} 인 네 개의 L자 모양 L_1, L_2, L_3, L_4로 나눈다.
3. 각 $i = 1, 2, 3, 4$에 대해 **채우기**$(L_i, n-1)$을 호출한다.



- 호출할 때마다 n이 1씩 작아지며, $n = 0$에서는 더 호출하지 않으므로 프로그램이 종료된다.

모듈러 연산

- 앞의 문제를 다시 보자.

문제 1

n이 주어진다. $1 + 2 + \cdots + n$의 값을 출력하여라.

- $n = 10^{12}$이면 답은 `long long int`의 범위마저 넘어간다.
- 이처럼 답이 아주 큰 문제도 존재한다.
- 이러한 문제를 쉽게 출제하고 해결할 수 있도록, PS에서는 "답을 998244353으로 나눈 나머지를 출력하여라"와 같은 조건을 사용하는 경우가 많다.

모듈러 연산

- 모듈러 연산은 어떤 상수 a로 나눈 나머지만을 계산하는 연산이다. a로 나눈 나머지를 modulo a에서의 값이라고 하며, a로는 보통 큰 소수를 사용한다.
- 두 수를 $x_1a + y_1$, $x_2a + y_2$라고 나타내면 다음이 성립한다.

$$(x_1a + y_1) + (x_2a + y_2) = (x_1 + x_2)a + (y_1 + y_2),$$
$$(x_1a + y_1) - (x_2a + y_2) = (x_1 - x_2)a + (y_1 - y_2),$$
$$(x_1a + y_1)(x_2a + y_2) = (x_1x_2a + x_1y_2 + x_2y_1)a + y_1y_2.$$

- 따라서 덧셈, 뺄셈, 곱셈은 각 수의 나머지만으로 계산해도, 전체를 계산한 뒤 나머지를 구한 것과 같은 결과를 얻는다.

모듈러 연산

- 나눗셈에서는 xy를 a로 나눈 나머지가 1이 되는 y를 x^{-1}로 생각한다.

$$xy \equiv 1 \pmod{a}$$

- 이때 y를 modulo a에서의 x의 **모듈러 역원**이라고 한다.
- a가 소수라면, modulo a에서 0이 아닌 모든 수는 모듈러 역원을 갖는다.
- 따라서 나눗셈을 역원의 곱셈으로 바꾸면 modulo a에서도 일반적인 사칙연산을 다룰 수 있다.

$$\frac{x}{y} \equiv xy^{-1} \pmod{a}$$

모듈러 연산

- 소수에 대한 모듈러 역원은 페르마의 소정리를 이용해 쉽게 구할 수 있다.

페르마의 소정리

소수 p와 p의 배수가 아닌 정수 a에 대해 다음이 성립한다.

$$a^{p-1} \equiv 1 \pmod{p}$$

- 따라서 $a \cdot a^{p-2} \equiv 1 \pmod{p}$이므로, a^{p-2}가 a의 모듈러 역원이다.
- 이제 a^{p-2}만 빠르게 구하면 모듈러 역원을 빠르게 구할 수 있다.
- 거듭제곱을 단순히 계산하면 $\mathcal{O}(p)$의 시간이 걸리므로, $p = 998244353$과 같이 큰 소수에서는 너무 느리다.

모듈러 연산

- 다음 두 식을 이용하면 지수를 매번 절반으로 줄일 수 있다.

$$a^{2k} = (a^k)^2, \qquad a^{2k+1} = a(a^k)^2$$

- 따라서 다음과 같이 a^{p-2}를 구하면 모듈러 역원을 $\mathcal{O}(\log_2 p)$의 시간에 계산할 수 있다.

```
long long pow(long long a, long long x, long long p) {
  if (x == 0) return 1;
  a %= p;
  long long tmp = pow(a, x / 2, p);
  tmp = tmp * tmp % p;
  if (x % 2) tmp = tmp * a % p;
  return tmp;
}

long long inverse = pow(a, p - 2, p);
```

모듈러 연산

- 페르마의 소정리의 증명이 궁금하다면 다음과 같이 생각할 수 있다. 그냥 받아들이고 사용해도 문제는 없다.
- 소수 p와 p의 배수가 아닌 정수 a를 고정하자.
- 서로 다른 $i, j \in \{1, 2, \dots, p-1\}$에 대해 $i \not\equiv j \pmod{p}$이므로 $ai \not\equiv aj \pmod{p}$이다.
- 따라서 $a, 2a, \dots, (p-1)a$를 p로 나눈 나머지는 순서만 뒤섞인 $1, 2, \dots, p-1$이다. 그러므로 이들을 모두 곱한 값도 modulo p에서 같다.

$$a^{p-1}(p-1)! \equiv (p-1)! \pmod{p}$$

- $(p-1)! \not\equiv 0 \pmod{p}$이므로 양변에서 이를 약분하면 $a^{p-1} \equiv 1 \pmod{p}$를 얻는다.
- 예를 들어 $p = 5$, $a = 3$이면 $3, 6, 9, 12$를 5로 나눈 나머지는 각각 $3, 1, 4, 2$이다.