Competitive Programming
Science Honors Program (SHP)
Session 4

Christian Yongwhan Lim

Saturday, October 24, 2026

Overview

- Greedy
- Dynamic Programming

Greedy

Greedy

- Locally optimum choice leads to global optimum!

Greedy

- Locally optimum choice leads to global optimum!

- Proving that a problem can be solved using greedy is the hard part!

Greedy

- Locally optimum choice leads to global optimum!

- Proving that a problem can be solved using greedy is the hard part!
  - Exchange argument;

Greedy

- Locally optimum choice leads to global optimum!

- Proving that a problem can be solved using greedy is the hard part!
  - Exchange argument;
  - Induction;

Exchange Argument: Concrete Steps

1. Define the greedy and optimal solutions.
2. Show that the solutions differ in some way.
3. Exchange moves between the solutions.
4. Prove that the exchange doesn't make the cost any worse.
5. Iterate the exchange until the greedy solution is reached.

Key Problem

- You are given n activities with their start and finish times.

- Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time.

Solution Idea?

- **Greedy!**

Solution Idea?

- **Greedy!**

- Pick the next activity whose **finish time** is the least among the remaining activities and the **start time** is more than or equal to the finish time of the previously selected activity.

Solution Idea?

- **Greedy!**

- Pick the next activity whose **finish time** is the least among the remaining activities and the **start time** is more than or equal to the finish time of the previously selected activity.

Why?

CSES 1090: Ferris Wheel

Solution Idea?

Solution Idea?

- Greedy
- Two Pointers

Solution

- https://cses.fi/paste/da882cbdbe41d7035569e2/

Dynamic Programming

- So, what is it?

Dynamic Programming

- So, what is it?

- Why is it infamous?

Dynamic Programming (Warm-Up)

- Fibonacci sequence

```
int f(int n) {
  if (n==0) return 0;
  if (n==1) return 1;
  return f(n-1) + f(n-2);
}
```

Dynamic Programming: Top-Down (Warm-Up)

- Fibonacci sequence

```
int f(int n) {
  if (n==0) return 0;
  if (n==1) return 1;
  return f(n-1) + f(n-2);
}
```

- **TOO SLOW!** How do you optimize this a bit?

Dynamic Programming: Top-Down (Warm-Up)

- One answer: **Memoization!**

```
const int MAXN = 100;
bool found[MAXN];
int memo[MAXN];
int f(int n) {
  if (found[n]) return memo[n];
  if (n==0) return 0;
  if (n==1) return 1;
  found[n] = true;
  return memo[n] = f(n-1) + f(n-2);
}
```

Dynamic Programming: Top-Down (Warm-Up)

- You may use `map` or `unordered_map` (though slower).

```
map<int, int> memo;
int f(int n) {
  if (memo.count(n)) return memo[n];
  if (n==0) return 0;
  if (n==1) return 1;

  return memo[n] = f(n-1) + f(n-2);
}
```

Dynamic Programming: Bottom-Up (Warm-Up)

- Another answer (to speed up): **bottom-up**

```
const int MAXN = 100;
int fib[MAXN];

int f(int n) {
  fib[0] = 0;
  fib[1] = 1;
  for (int i = 2; i <= n; i++)
    fib[i] = fib[i-1] + fib[i-2];
  return fib[n];
}
```

Dynamic Programming: Bottom-Up (Warm-Up)

- To save a memory, since you are using only previous two values, you can do:

```
const int MAX = 3;
int fib[MAX];
int f(int n) {
  fib[0] = 0;
  fib[1] = 1;
  for (int i = 2; i <= n; i++)
    fib[i%MAX] = fib[(i-1)%MAX] + fib[(i-2)%MAX];
  return fib[n%MAX];
}
```

Classic Dynamic Programming (DP) Problems

- Counting all possible paths from top left to bottom right corner of a matrix (Combinations)
- 0-1 Knapsack

Path Counting


BREAK

Group Exercise

- You may work in groups with those sitting near you.
- Each person must submit their own solutions.
- You have **75 minutes**.
- Afterward, you will present your solution!


THANK YOU

0-1 Knapsack