Introduction to Algorithms

Science Honors Program (SHP)

Session 4

Christian Lim

Saturday, March 9, 2024

Slide deck in github

- You may get to the link by:
  - https://github.com/yongwhan/
  - => yongwhan.github.io
  - => columbia
  - => shp
  - => session 4 slide

Overview

- **Divide and Conquer**
- Break #1 (5-minute)
- **Greedy**
- Break #2 (5-minute)
- **Dynamic Programming**
- *Break #3 (5-minute)
- ***Interactive Session**

- ***: only if there is time at the end!**

CodeForces Columbia SHP Algorithms Group

- While I take the attendance, please join the following group: **https://codeforces.com/group/lfDmo9iEr5**

- We will be using them in the last portion of the session today!

Attendance

- Let's take a quick attendance before we begin!

Divide and Conquer

- **Divide** the problem into multiple subproblems,

Divide and Conquer

- **Divide** the problem into multiple subproblems,

- **Conquer** each subproblem,

Divide and Conquer

- **Divide** the problem into multiple subproblems,
- **Conquer** each subproblem,
- **Combine** the result!

Binary Exponentiation

- Calculate $\mathbf{a^n}$ fast!

Binary Exponentiation: Main Idea

$$a^n = \begin{cases} 1 & \text{if } n == 0 \\ \left(a^{\frac{n}{2}}\right)^2 & \text{if } n > 0 \text{ and } n \text{ even} \\ \left(a^{\frac{n-1}{2}}\right)^2 \cdot a & \text{if } n > 0 \text{ and } n \text{ odd} \end{cases}$$

Binary Exponentiation: Implementation

```
long long binpow(long long a, long long b) {
    if (b == 0)
        return 1;
    long long res = binpow(a, b / 2);
    if (b % 2)
        return res * res * a;
    else
        return res * res;
}
```

Merge Sort

- Sort an array of integers in ascending order.

- Time: O(n log n)
- Space: O(n)

Merge Sort: Implementation (divide & conquer)

```
// vector<int> v, aux;
void merge_sort(int left, int right) {
  if (left==right) return;
  int middle=(left+right)/2;
  merge_sort(left,middle),
  merge_sort(middle+1,right);
```

- Time: O(n log n)
- Space: O(n)

Merge Sort: Implementation (combine)

```
    aux.clear();
    int i=left, j=middle+1;
    while (i<=middle || j<=right)
      if (j>right || (i<=middle && v[i]<v[j]))
        aux.push_back(v[i]), i++;
      else aux.push_back(v[j]), j++;
    for (int i=left; i<=right; i++)
      v[i]=aux[i-left];
  }
```

Fast Fourier Transform (FFT) and Karatsuba Algorithm

- Multiply two polynomials in **n log n** time (instead of naive n^2)!
- **Karatsuba** is similar, but works well for multiplying numbers!
- We will cover **FFT** and **Karatsuba** when we jump to math section later!

Practice Problems

- **Divide and Conquer** (available in CodeForces group!)
  - https://codeforces.com/problemset/problem/1490/D
  - https://codeforces.com/problemset/problem/1741/D
  - https://codeforces.com/problemset/problem/1167/B
  - https://codeforces.com/problemset/problem/1385/D
  - https://codeforces.com/problemset/problem/1373/D
  - https://codeforces.com/problemset/problem/559/B
  - https://codeforces.com/problemset/problem/459/D
  - https://codeforces.com/problemset/problem/448/C
  - https://codeforces.com/problemset/problem/321/C


BREAK #1

Greedy Algorithm

- **Locally optimum choice leads to global optimum!**

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

Question #1

- 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.

Discuss for few minutes!

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?

Question #2

- Balanced strings are those that have an equal quantity of 'L' and 'R' characters.
- Given a **balanced** string s (so, to be explicit, it only contains 'L' and 'R' only), split it in the maximum amount of balanced strings.
- Return the maximum amount of split balanced strings.

Discuss for few minutes!

Model Solution

- Time: O(n)
- Space: O(1)

```
int balancedStringSplit(const string &s) {
  int ret=0, cur=0;
  for (char ch : s) {
    if(ch=='R') cur++;
    else cur--;
    if(!cur) ret++;
  }
  return ret;
}
```

Question #3

- A string S of lowercase English letters is given.
- We want to partition this string into as many parts as possible so that each letter appears in at most one part, and return a list of integers representing the size of these parts.

Discuss for few minutes!

- Time: O(n log n)
- Space: O(1)

Model Solution

```
vector<int> partitionLabels(const string &S) {
  vector<int> ret;
  int prev=-1, mx=0, n=S.size();
  map<char,int> mp;
  for (int i=0; i<n; i++)
    mp[S[i]]=i;
  for (int i=0; i<n; i++) {
    mx=max(mp[S[i]],mx);
    if(i==mx) ret.push_back(i-prev), prev=i;
  }
  return ret;
}
```

Reference

- **For more details, please take a look at the following tutorials:**
  - https://en.wikipedia.org/wiki/Greedy_algorithm
  - https://www.geeksforgeeks.org/greedy-algorithms

- **Also, check out the following CodeForces Problem Set:**
  - https://codeforces.com/problemset?tags=greedy

Practice Problems

- **Greedy** (available in CodeForces group!)
  - https://codeforces.com/problemset/problem/1538/A
  - https://codeforces.com/problemset/problem/1496/A
  - https://codeforces.com/problemset/problem/1485/A
  - https://codeforces.com/problemset/problem/1428/C
  - https://codeforces.com/problemset/problem/1355/B
  - https://codeforces.com/problemset/problem/1873/F
  - https://codeforces.com/problemset/problem/1624/D
  - https://codeforces.com/problemset/problem/274/A
  - https://codeforces.com/problemset/problem/1437/D


BREAK #2

Dynamic Programming

- **Save the repeated computations in memory to avoid computing them again!**

- **Common Types**:
  - 1D;
  - 2D;
  - Bitmask;
  - Tree;

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);
}
```

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 Problems

- **0-1 Knapsack**
- **Subset** Sum
- Longest Increasing Subsequence (**LIS**)
- Counting all possible paths from top left to bottom right corner of a matrix (**Combinations**)
- Longest Common Subsequence (**LCS**)
- Longest Path in a Directed Acyclic Graph (**DAG**)
- Coin Change (**CC**)
- Rod Cutting
- Edit Distance (**Levenshtein**)

Classic Dynamic Programming Problems

- **"Broken Profile"**
  - A number of ways to fill a grid with dominoes.
- **Interval**
  - A minimum number of insertions to make a string palindrome.
- **Bitmask**
  - Traveling Salesman Problem (TSP)
- **Digit**
- **Tree**
- Longest Palindromic Subsequence (**Manacher**)
- **We will revisit these topics throughout the sessions!**

Simple Game

- Alice and Bob take turns playing a game, with Alice starting first. Initially, there is a number N on the chalkboard. On each player's turn, that player makes a move consisting of:
  - Choosing any proper divisor x of N.
  - Replacing the number N on the chalkboard with N - x.
- Also, if a player cannot make a move, they lose the game.
- Return true if and only if Alice wins the game, assuming both players play optimally.

Discuss for few minutes!

Solution Idea?

- 1D DP

Model Solution

- Time: $O(N^2)$
- Space: $O(N)$

```
const int mx=1007;
bool divisorGame(int N) {
  vector<bool> dp(mx,false);
  for (int n=2; n<mx; n++) {
    for (int x=1; x<n; x++)
      if(n%x==0&&!dp[n-x]) {
        dp[n]=true; break;
      }
  }
  return dp[N];
}
```

Longest Common Subsequence

- Given two strings s and t, return the length of their longest common subsequence.
- A subsequence of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters. (e.g., "ace" is a subsequence of "abcde" while "aec" is not).
- A common subsequence of two strings is a subsequence that is common to both strings. If there is no common subsequence, return 0.

Discuss for few minutes!

Solution Idea?

- 2D DP

- Time: O(nm)
- Space: O(nm)

Model Solution

```
int longestCommonSubsequence(const string &s,
                             const string &t) {
  int n=s.size(), m=t.size();
  vector<vector<int>> dp(n+1, vector<int>(m+1,0));
  for (int i=0; i<n; i++)
    for (int j=0; j<m; j++)
      dp[i+1][j+1]=(s[i]==t[j]) ? dp[i][j]+1
                   : max(dp[i+1][j],dp[i][j+1]);
  return dp[n][m];
}
```

Hamiltonian Flight (CSES 1690)

- There are n cities and m flight connections between them. You want to travel from Syrjälä to Lehmälä so that you visit each city exactly once. How many possible routes are there?

Discuss for few minutes!

Solution Idea?

- **Bitmask DP**

- Let's cover the exact implementation details later!

Tree Matching (CSES 1130)

- A matching is a set of edges where each node is an endpoint of at most one edge. What is the maximum number of edges in a matching?

Discuss for few minutes!

Solution Idea?

- **Tree DP**

- Let's cover the exact implementation details later! (Once we cover flow)

Reference

- **For more details, please take a look at the following tutorials:**
  - https://usaco.guide/gold/dp-bitmasks?lang=cpp;
  - https://usaco.guide/gold/dp-trees?lang=cpp;

- **Also, check out the following CodeForces Problem Set:**
  - https://codeforces.com/problemset?tags=dp;

Practice Problems

- **Dynamic Programming** (available in CodeForces group!)
  - https://codeforces.com/problemset/problem/1566/C
  - https://codeforces.com/problemset/problem/919/B
  - https://codeforces.com/problemset/problem/522/A
  - https://codeforces.com/problemset/problem/1037/C
  - https://codeforces.com/problemset/problem/1108/D
  - https://codeforces.com/problemset/problem/1389/C
  - https://codeforces.com/problemset/problem/1288/C
  - https://codeforces.com/problemset/problem/1091/D
  - https://codeforces.com/problemset/problem/645/D


BREAK #3

Again, CodeForces Columbia SHP Algorithms Group

- Please join the following group: **https://codeforces.com/group/lfDmo9iEr5**

No Class Next Week! Graph on March 23!

- Due to Spring Break, there will be **no class** on March 16!
- On March 23, we will cover graph algorithms:
  - **Shortest Paths**
  - **Minimum Spanning Trees**
  - **Lowest Common Ancestor**
  - **Flows**

Slide Deck

- You may **always** find the slide decks from:
  - https://github.com/yongwhan/yongwhan.github.io/blob/master/columbia/shp


THANK YOU