Pattern search problem

- Given a string S and pattern P, what are the positions where P appears?
  - Naive O(NM) where N = len(S) and M = len(P)
  - What about **hashing** substrings?
  - If we can **hash a substring**, then it is a **single value**
  - So we can hash the pattern = patCode
  - Search S for substrings that has **same** a hash value?
  - Can we do that **efficiently**? **Rabin–Karp algorithm**
  - E.g. can we get 2nd substring hash from 1st hash in O(1)?
  - We will **revise** some concepts to make things easier
- Many problems can be solved using same idea

Recall: Fixed size sliding window

- Given an array of N values, find M consecutive values that has the max sum?
- A bruteforce to compute that is just O(NM) by starting from every index and compute M values. Matter of 2 nested loops
- Observation: What is the relation between the first M values and 2nd M values?

Recall: Fixed size sliding window

- Let A = {1, 2, 3, 4, 5, 6, -3}, M = 4
  - 1st M values = **1**+2+3+4 = 10
  - 2nd M values = 2+3+4+**5** = 10**-1+5** = 14
  - 3rd M values = 3+4+5+6 = 14-2+6 = 18
  - 4th M values = 4+5+6-3 = 18-3-3 = 12
  - So answer is max(14, 18, 12) = 18
- We create a **window** of **fixed** size M
  - cur window = last window - its first item + new mth item
- This is also called **rolling** sum
  - E.g. you need ***12 month rolling sum*** on your monthly expenses (the sum of last 12 months for every month)

Recall: Number Bases

- Base 10 (Decimal) is the usual base in our life
- Base 2 (Binary) is in computer world (on/off)
- Base 8 (Octal): Christmas is Halloween?
- Base 16 (Hexadecimal) with A=10...F=15
- Base K has K digits: 0, 1, 2….K-1
- We can represent any base!
  - Up to 36, you can use Alphanumeric (A-Z), 26 letters
  - X = Tn*b^n+…..+ T2*b^2 +T1*b^1+T0*b^0 (base b)
  - 5736 = 5*10^3 + 7*10^2 + 3*10^1 + 6 * 10^0 (base 10)

Recall: Number Bases

- 5736 = 5*10^3 + 7*10^2 + 3*10^1 + 6 * 10^0
- 5736 = ((5*10+7)*10+3)*10+6
- The above format is actually a **polynomial** with coefficients are 5, 7, 3, 6 and base a = 10

$$x_0 a^{k-1} + x_1 a^{k-2} + \ldots + x_{k-2} a + x_{k-1}$$

- What we do in base 10 is same for other bases
  - 2AF3 = 2*16^3 + A * 16^2 + F * 16^1 + 3 * 16^0

Recall: Hash Function

- For simplicity, it maps an object (string, array or a structure) to a single value
- One popular use is hash table, where we hash items to a single value < M
  - It can be used for fast lookup where similar hash values objects can be grouped together.
- Polynomial Hash Function
  - Express the object as polynomial, compute sum % M
  - It might avoid collisions (same hash value for differnet objects)

Recall: Hash Function

- Let's hash array {15, 137, 12, 120}
  - Let's assume polynomial a = 19 (if a = 10, base 10)
  - Let's use mode M = 300
  - [(((15*19) + 137)*19+12)*19+120]%300
  - You can of that as shift left/add operations
  - Add 15, left shift
  - Add 137, left shift
  - Add 12, left shift
  - Add 120
  - We shouldn't wait tell last step to make mode or result will overflow. So apply the mod after each operation

Pattern search problem

- Given a string S and pattern P, what are the positions where P appears?
  - Seems similar to the **consecutive max sum**, but in sum case we can get the next sum in O(1)!
  - Can we adapt this idea based on hashing to have a number for set of characters/values?
  - E.g. represent each substring of length M using hash values and do the comparison on values **not substrings**?

Rolling Hash

- Instead of a direct **rolling sum**, we will do a **rolling hash**
- That is, we compute hash value of first M characters, then next M characters and so on
- Can we get 2nd hashing of a substring in O(1) from the first one? Generally, next from last
- Yes, use **polynomial** hash function. Then we can add and remove terms pretty **easy & fast**

String Polynomial Hash Function

```
#define MOD 2000000011ll
#define BASE 53ll

ll getHashValue(string pat) {
  ll patCode = 0;
  for (int i = 0; i < (int) pat.size(); ++i) {
    patCode = patCode * BASE; // left shift
    patCode += pat[i];        // add value
    patCode %= MOD;           // mod to avoid overflow
  }
  return patCode;
}
```

Trace string abcd

You will notice we build it as if is reversed: **dcba**

As we add a, then left shift. So eventually a is 4 items shifted. It is now base^3

14721698 dcba

Polynomial Hashing utilities

- It seems if we are manipulating several sub-strings, we will keep doing the same operations copy and paste
- Let’s define utilities that make coding easier
- We simply need to
  - Shift left (15 shift left base 10 = 150)
  - Add value at some index (150 add 7 = 157)
  - Let’s write a generic code that helps in several problems

Polynomial Hashing utilities

```
ll fastpow(ll num, ll p) {
  if (p == 0)
    return 1;
  if (p % 2)
    return (num % MOD * fastpow(num, p - 1)) % MOD;
  ll a = fastpow(num, p / 2);
  return (a * a) % MOD;
}

ll removeAt(ll code, int idx, int val) {
  return (code - (val * fastpow(BASE, idx)) % MOD + MOD) % MOD;
}

ll addAt(ll code, int idx, int val) {
  return (code + (val * fastpow(BASE, idx)) % MOD) % MOD;
}

ll shiftLeft(ll code) {
  return (code * BASE) % MOD;
}
```

String Polynomial Hash Function

```
ll getHashValue2(string pat) {
  ll patCode = 0;
  for (int i = 0; i < (int) pat.size(); ++i) {
    patCode = shiftLeft(patCode);
    patCode = addAt(patCode, 0, val(pat[i]));
  }
  return patCode;
}
```

Rolling Hash Pattern Search

- Compute hash value for the pattern = patCode
- Compute hash value for first M letters
  - is strCode == patCode? If yes, then **potential** match
- Compute hash value of 2nd M letters
  - In O(1) compute the new hash value
  - Remove a position and add another
  - is strCode == patCode? If yes, then **potential** match
- Compute 3rd M letters ….and so on

Rolling Hash Pattern Search

- Assume input string abcde and M = 4
  - So first M letters = **abcd**
  - Second M letters bcde = remove a and add e
- Our function builds it reversed
  - So we have now **bcda** built, with a represents **highest** idx
  - code = (((a * base)+b)*base+c)*base+d **Or**
  - code = a x base^3 + b x base^2 + c x base + d
  - To remove a, we need to remove it under power M-1
  - new code = code - a x base^3 + e

Rolling Hash Pattern Search

```
void pattern_search(string main, string pat) {
  int n = pat.size();
  ll patCode = 0;
  for (int i = 0; i < (int) pat.size(); ++i) {
    patCode = shiftLeft(patCode);
    patCode = addAt(patCode, 0, val(pat[i]));
  }

  ll subCode = 0;
  string subStr;
  for (int i = 0; i < (int) main.size(); ++i) {
    if (i - n >= 0) {
      subCode = removeAt(subCode, n - 1, main[i - n]);
      subStr.erase(subStr.end()-1);
    }
    subCode = shiftLeft(subCode);
    subCode = addAt(subCode, 0, main[i]);
    subStr.insert(subStr.begin(), main[i]);
    if(patCode == subCode)
      cout << subCode << "\t" << subStr<<"\n";
  }
}
```

```
pattern_search("abcd..abcd...abcd",
                "abcd");

14721698  dcba
14721698  dcba
14721698  dcba
```

Removing at the begin

- So far, we were removing the first added letter
- E.g. for string abcd
  - It is added as dcba, where a has base power = 3
- What if we want to do removing for letter in begin?
  - We added it by 2 operations
  - Shift left
  - Add at position 0
  - To remove it:
  - Remove at position 0, then **shift right**?

Removing at the begin

- Let X = 4578
  - X = 4*1000 + 5*100 + 7*10 + 8
- Remember our target remove operation **minimizes** the length
- How to remove the 4? Subtraction
  - X -= 4*1000 => X = 578
- How to remove the 8? Subtraction not enough
  - X -= 8 => 4570 NOT 457
  - We need also to divide by 10 (or the base)
  - X (4570) / 10 = 457 (YES: So **remove, then divide**)

Dividing under mod

- How to do the **shift right** (Division)?
- This is called Modinverse (see math playlist)
- For now, 30 / 5 = 30 * ⅕= 6
- We need to do 4570 / Base
  - So = 4570 * 1/Base but under MOD
- If Base is prime,
  - BaseInv = 1/Base = Pow(Base, Mod-2) % MOD
- Then 4570 / Base = 4750 * BaseInv

Division (shift right utility)

```
#define MOD 200000001ll
#define BASE 53ll
#define BASE_INV 128301887ll    // pow(BASE, MOD-2)%MOD

ll shiftRight(ll code) {
  return (code * BASE_INV) % MOD;
}
```

Longest palindromic suffix

- [UVA 11475]. Let's' try one more example
  - Given string: produce the smallest palindrome that can be formed by **adding** zero or more characters at its **end**.
  - abba => abba
  - zyabba => zyabba**yz**
- Observation
  - For zyabba => abba is the longest suffix that is palindrome. So we need to append zy reversed
  - So actual task is, given a string what is the lowest suffix **index** that is palindrome

Longest palindromic suffix

- Brute Force
  - Start from the end, keep building a suffix and its reverse
  - For every suffix = reverse => we have **potential** one
  - E.g. **zyabba**
  - a vs a [ok]
  - ba vs ab
  - bba vs abb
  - abba vs abba [ok, actually longest]
  - yabba vs abbay
  - zyabba vs abbayz
- For faster comparison, use roll hashing

Longest palindromic suffix

```
// return lowest suffix index that is palindrome
int longestSuffixPalindrome(string str) {
  int n = str.size(), longestSuffix = 0;
  ll strCode = 0, strRevCode = 0;

  // start from end, find longest suffix = reverse of suffix
  for (int i = n - 1, len = 0; i >= 0; --i, ++len) {
    strCode = shiftLeft(strCode);
    strCode = addAt(strCode, 0, str[i]);
    //Note, next is not so efficient, as we compute pow each step
    strRevCode = addAt(strRevCode, len, str[i]);

    if (strCode == strRevCode)
      longestSuffix = n - i;
  }
  return longestSuffix;
}
```

Longest palindromic suffix

```
int main() {
#ifndef ONLINE_JUDGE
  freopen("test.txt", "rt", stdin);
#endif

  string str;
  while (getline(cin, str)) {

    int longestSuffix = longestSuffixPalindrome(str);

    cout<<str;
    // print the first few letters reversed
    for (int i = (str.size() - longestSuffix) - 1; i >= 0; i--)
      cout<<str[i];
    cout<<"\n";
  }
  return 0;
}
```

Handling Collisions

- In fact, there is a clear bug in all what we did
- When we hash 2 different strings, they may have the same value
- So code will be cheated and think they are same substrings, while they are not!
- How to handle? In competitions? In real life?

Handling Collisions

- Just assume you will be lucky :)
  - Most probably you will get AC. Judges doesn't have specific test case for your prime base / MOD
  - If got WA: Just try another base/Mod and resubmit :D
  - In TC/CF hacks, one can invent specific test case to get your code down
- Double Or tripple hashing
  - Instead of 1 hash value for a substring, maintain 2
  - It is much harder that a test case will be designed for that
- Actual comparison (100% safe, but slow)
  - Just do actual comparisons for your strings in matches

To sum up

- Matter of efficient brute force!
- Whenever you need to generate substrings, then polynomial hashing might be a trivial to go.
  - Many Times other harder algorithms can be applied such as KMP or suffix arrays
  - But this one doesn't need more thinking! Just efficient BF
- Readings: Please read/think in the examples here