Bài 1: House Robber (Medium)

https://leetcode.com/problems/house-robber/

Cho 1 mảng các số nguyên, yêu cầu lấy ra các số trong mảng đấy sao cho tổng là lớn nhất, với điều kiện không được lấy 2 phần tử cạnh nhau trong mảng.

Ví dụ: A = [1, 2, 3, 1]

Output: 4. Lấy phần tử đầu tiên và thứ 3 của mảng sẽ cho tổng lớn nhất.

Problem analysis using brute-force:

- How to generate candidate and how to check candidate: Tạo ra tất cả các cách lấy số sao cho không lấy 2 số liên tiếp, sau đó tìm tổng của mảng thu được lớn nhất.
- Memory space, execution time and data structure:
  - Memory space: O(2^N)
  - Execution time: O(N)
  - Data structure: Array

Brute-force code:

```
vector<vector<int>> ret;
void dfs(vector<int> a, vector<int>& temp, int index) {
	if (index >= a.size()) {
		ret.push_back(temp);
	}
	for (int i=index; i<a.size(); ++i) {
		temp.push_back(a[i]);
		dfs(a, temp, i+2);
		temp.pop_back();
	}
}

int main(){
	vector<int> a{1, 2, 3, 1};
	vector<int> temp;
	dfs(a, temp, 0);
	int ans = 0;
	for (int i=0; i<ret.size(); ++i) {
```

```
            int count = 0;
            for (int j = 0; j<ret[i].size(); ++j) count+=ret[i][j];
            ans = max(ans, count);
        }
        cout<<ans;
        return 0;
    }
```

Improvement analysis by using dynamic programming algorithm:

- Theoretical basis to apply dynamic progamming: Khi duyệt đến 1 phần tử trong mảng, ta chỉ có 2 lựa chọn là lấy hoặc không lấy. Nếu lấy thì giá trị lấy được là A[i] = A[i] + A[i-2], nếu không lấy giá trị thu được là A[i] = A[i-1]. Từ đây ta có công thức: dp[i] = max(A[i] + dp[i-2], dp[i-1]).
- Memory space, execution time and data structure.
    - Memory space: O(N)
    - Execution time: O(N)
    - Data structure: Array
- Code

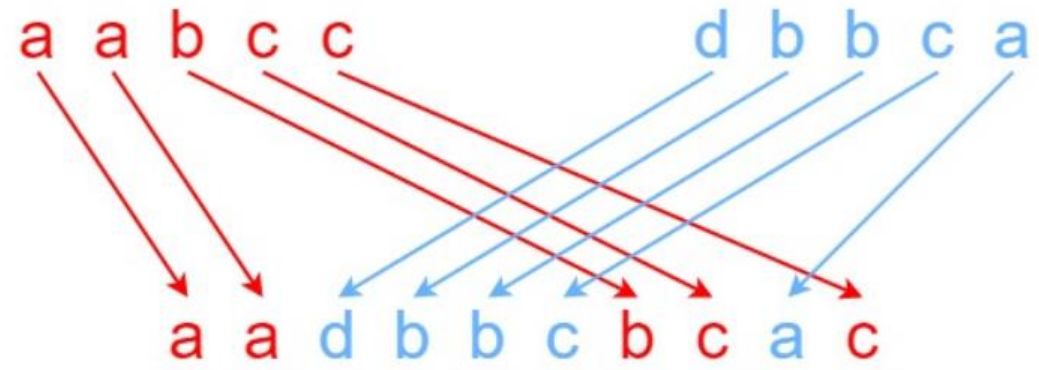
```
int Solve(vector<int>& a) {
    if (a.size() == 1) return a[0];
    a[1] = max(a[1], a[0]);
    for (int i=2; i<a.size(); ++i) {
        a[i] = max(a[i] + a[i-2], a[i-1]);
    }
    return a[a.size()-1];
}
```

Bài 2: Interleaving String (Medium)

https://leetcode.com/problems/interleaving-string/

Cho 3 chuỗi s1, s2, s3. Kiểm tra xem có thể tạo được chuỗi s3 từ việc phân tách và kết hợp 2 chuỗi s1 và s2.

Example 1:




```
Input: s1 = "aabcc", s2 = "dbbca", s3 = "aadbbcbcac"
Output: true
```

Giới hạn: 0 <= s1.length, s2.length <= 100

0 <= s3.length() <= 200

Problem analysis using brute-force:

- How to generate candidate and how to check candidate: Sử dụng brute-force để tạo ra tất cả các chuỗi mới từ cách kết hợp 2 chuỗi ban đầu, nếu chuỗi tạo ra giống chuỗi s3 thì trả về true, ngược lại trả về false.
- Memory space, execution time and data structure:
  - Memory space: O(s3.length())
  - Execution time: O(2^(s1.length() + s2.length()))
  - Data structure: Array

Brute-force code:

```
vector<string> ret;
string s1, s2, s3;
void dfs(int x, int y, string& temp) {
        if (temp.length() == s3.length()) {
                ret.push_back(temp);
                return;
        }
        if (x < s1.length()) {
                temp += s1[x];
                dfs(x+1, y, temp);
                temp.pop_back();
```

```
        }

        if (y < s2.length()) {
            temp += s2[y];
            dfs(x, y+1, temp);
            temp.pop_back();
        }
    }

    int main(){
        s1 = "aabcc";
        s2 = "dbbca";
        s3 = "aadbbbaccc";
        string temp = "";
        dfs(0, 0, temp);
        for (int i=0; i<ret.size(); ++i) cout<<ret[i]<<endl;
        int ans = 0;
        return 0;
    }
```

Improvement analysis by using dynamic programming algorithm:

- Theoretical basis to apply dynamic programming:

Trong phương pháp sử dụng brute-force, có nhiều bài toán bị lặp lại dẫn đến thời gian tính toán lớn.

Để tối ưu bằng phương pháp quy hoạch động, ta xét tại vị trí x của chuỗi s1 và y của chuỗi s2: nếu s1[x] = s3[x+y] thì ta chọn s1[x] cho vị trí s3[x+y] và giải bài toán nhỏ hơn ở vị trí x-1 và y. Tương tự cho vị trí y của s2.

dp[x][y] = (dp[x-1][y] & (s3[x+y-1] == s1[x-1]) | (dp[x][y-1] & (s3[x+y-1] == s2[y-1])

- Memory space, execution time and data structure.
    - Memory space: O(s1.length() * s2.length())
    - Execution time: O(s1.length() * s2.length())
    - Data structure: 2D-Array
- Code

```
vector<vector<int>> dp;

int recur(string& s1, string& s2, string& s3, int i, int j) {
    if (i < 0 || j < 0) return 0;
    if (dp[i][j] != -1) return dp[i][j];
    int a = 0, b = 0;
    if (i > 0 && s1[i-1] == s3[i+j-1]) {
        if (recur(s1, s2, s3, i-1, j)) a=1;
    }
    if (j > 0 && s2[j-1] == s3[i+j-1]) {
        if (recur(s1, s2, s3, i, j-1)) b =1;
    }
    dp[i][j] = a | b;
    return dp[i][j];
}

bool Solve(string s1, string s2, string s3) {
    int m = s1.length(), n = s2.length();
    if (m+n != s3.length()) return false;
    dp = vector<vector<int>> (s1.length()+1, vector<int> (s2.length()+1, -1));
    dp[0][0] = 1;
    return (recur(s1, s2, s3, m, n) > 0) ? true : false;
}
```

Bài 3: Add One (Medium)

https://codeforces.com/problemset/problem/1513/C

Cho 1 số nguyên dương n, có m thác tác thực hiện lên số nguyên dương này trong đó mỗi thao tác sẽ thay thế mọi chữ số d trong số n bằng d+1.

Ví dụ: n = 1912, m = 2

Thao tác 1: thay thay 1 bằng 2, 9 bằng 10, 2 bằng 3 tác có n = 21023

Thao tác 2: thay 2 bằng 3, 1 bằng 2, 0 bằng 1, 2 bằng 3, 3 bằng 4 ta có n = 32134

In ra số ký tự của số n thu được sau khi thực hiện m thao tác, số chữ số này có thể rất lớn, nên in ra theo phần dư của nó khi chia cho 10^9 + 7

Đầu vào: t testcase (1 <= t <= 2*10^5), n (1<=n <= 10^9), m(1<=m<=2*10^5)

Đầu ra: mỗi testcase in ra số chữ số của số thu được

Problem analysis using brute-force:

- How to generate candidate and how to check candidate: Lưu các chữ số của n vào 1 vector, sau mỗi thao tác tạo ra 1 số mới và cập nhật các chữ số này vào vector
- Memory space, execution time and data structure:
  - Memory space: O(2^(M/9))
  - Execution time: O(2^(M/9))
  - Data structure: Array

Brute-force code:

```
int Solve(vector<int>& digits, int m) {
    vector<int> digits_2;
    for(int i = 0; i < m; i ++){
        for(int j = 0; j<digits.size(); j++){
            int d = digits[i];
            if(d == 9) {
                digits_2.push_back(1);
                digits_2.push_back(0);
            }
            else{
                digits_2.push_back(d+1);
            }
        }
        digits.clear();
        for(int j = 0; j < digits_2.size(); j++){
            digits.push_back(digits_2[i]);
        }
    }
}
```

Improvement analysis by using dynamic programming:

- Theoretical basis to apply binary search or divide and conquer: đặt dp[i] là số chữ số sau khi áp dụng i thao tác lên 10 ta có
  - dp[i] = 2 (i thuộc [0,8])
  - dp[i] = 3 (i thuộc [9,9])
  - dp[i] = dp[i-9] + dp[i-10] (i > 9)

ta tính và lưu mảng dp[i] với i thuộc [0,2*10^5]

với số n bất kỳ, phân tích ra các chữ số khác nhau và với mỗi chữ số ta biến đổi chữ số d này thành 10 sau (10 - d) thao tác.

- Memory space, execution time and data structure.
  - Memory space: O(M)
  - Execution time: O(M)
  - Data structure: Array
- Code

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

const int max_n = 200005, mod = 1000000007;
long long dp[max_n];

int main(){

    for(int i=0; i<9; i++) dp[i] = 2;
    dp[9] = 3;
    for(int i=10; i<max_n; i++){
        dp[i] = (dp[i-9] + dp[i-10])%mod;
    }

    int t;
    scanf("%d",&t);
    while(t--){
        int n, m;
        scanf("%d%d",&n,&m);
        long long ans = 0;
        while(n > 0){
            int x = n%10;
            ans += ((m + x < 10) ? 1 : dp[m + x - 10]);
            ans %= mod;
            n/=10;
        }
        printf("%lld\n",ans);
    }
}
```

Bài 4: String Reversal (Hard)

https://codejam.lge.com/problem/19597

Cho N strings, với mỗi string trong dãy có thể chọn đảo ngược string hoặc gữ nguên string, mỗi phép đảo ngược cho bit 1, không đảo ngược cho bit 0.

Tìm chuỗi bít có giá trị nhỏ nhất tương ứng để biến dãy N strings thành dãy có thứ tự tăng dần theo từ điển

VD: N = 3, and S[1] = "ABC", S[2] = "XC", and S[3] = "DZ"

Chuỗi bit: bit[1] = 0, bit[2] = 0; bit[3] = 0 ta có S[1] = "ABC", S[2] = "XC", and S[3] = "DZ" thỏa mãn tăng dần theo từ điển

Input:

Dòng đầu tiên là số lượng test case T (1 ≤ T ≤ 50)

Với mỗi test case dòng đầu tiên là 1 số nguyên ứng với N

N dòng tiếp theo tương ứng với chuỗi s[i] (2 <= length s[i] <= 20)

Output:

In chuỗi bit có giá trị nhỏ nhất sao tương với với chuỗi thao tác để biến n chuỗi s[i] tăng dần theo thứ tự từ điển

Problem analysis using brute-force:

- How to generate candidate and how to check candidate: tăng dần chuỗi bit bits[N], và kiểm tra tính hợp lệ, nếu hợp lệ thì dừng lại, in chuỗi bit thu được
- Memory space, execution time and data structure:
    - Memory space: O(N)
    - Execution time: O(2^N)
    - Data structure: Array

Brute-force code: NA

Improvement analysis by using binary search or divide and conquer algorithm:

- Theoretical basis to apply binary search or divide and conquer: cost[i][0] là hàm giá nhỏ nhất khi chọn chuỗi thứ i không nghịch đảo, cost[i][1] là hàm giá nhỏ nhất khi chọn chuỗi thứ i nghịch đảo.

cost[i][0] = min(cost[i-1][0]*2,cost[i-1][0]*2)

cost[i][1] = min(cost[i-1][1]*2,cost[i-1][1]*2)

- Memory space, execution time and data structure.
    - Memory space: O(N)
    - Execution time: O(N)
    - Data structure: Array
- Code

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

int compareString(string &s1, string &s2, bool isRevertSt1, bool
isRevertSt2){
    int st1Len = s1.size();
```

```
    int st2Len = s2.size();
    int minlen = min(st1Len,st2Len);

    int results = -1;
    for(int i = 0; i < minlen; i++){
        char ch1, ch2;
        ch1 = isRevertSt1 ? s1[st1Len - i - 1] : s1[i];
        ch2 = isRevertSt2 ? s2[st2Len - i - 1] : s2[i];
        if(ch1 < ch2) return -1;
        if(ch1 > ch2) return 1;
    }
    if(st1Len<st2Len) return -1;
    if(st1Len>st2Len) return 1;
    return 0;
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);

    int T;
    cin >> T;
    while (T--) {

        int N;
        string ss[150];
        cin >> N;
        for(int i = 0; i < N; i++){
            cin >> ss[i];
        }
        int back0[155] = {0};
        int back1[155] = {0};
        int cost0[155] = {0};
        int cost1[155] = {0};
        cost1[0] = 1;

        int cmp0, cmp1, cmp2, cmp3;
        for(int i = 1; i<N; i++){
            cmp0 = compareString(ss[i-1],ss[i],0,0);
            cmp1 = compareString(ss[i-1],ss[i],1,0);
            cmp2 = compareString(ss[i-1],ss[i],0,1);
            cmp3 = compareString(ss[i-1],ss[i],1,1);

            if(cmp0 < 0 && cmp1 < 0){
                if(cost0[i-1] < cost1[i-1]){
                    back0[i] = 0;
                    cost0[i] = cost0[i-1];
                }
                else {
                    back0[i] = 1;
                    cost0[i] = cost1[i-1];
                }
            }
            else if(cmp0 < 0 && cost0[i-1] != 1000){
                back0[i] = 0;
                cost0[i] = cost0[i-1];
```

```
            }
            else if(cmp1 < 0 && cost1[i-1] != 1000) {
                back0[i] = 1;
                cost0[i] = cost1[i-1];
            }
            else {
                back0[i] = 0;
                cost0[i] = 1000;
            }

            if(cmp2 < 0 && cmp3 < 0){
                if(cost0[i-1] < cost1[i-1]){
                    back1[i] = 0;
                    cost1[i] = cost0[i-1];
                }
                else {
                    back1[i] = 1;
                    cost1[i] = cost1[i-1];
                }
            }
            else if(cmp2 < 0 && cost0[i-1] != 1000){
                back1[i] = 0;
                cost1[i] = cost0[i-1];
            }
            else if(cmp3 < 0 && cost1[i-1] != 1000){
                back1[i] = 1;
                cost1[i] = cost1[i-1];
            }
            else {
                back1[i] = 0;
                cost1[i] = 1000;
            }

            if(cost0[i] == cost1[i]) {
                cost0[i] = 0;
                cost1[i] = 1;
            }
        }

        int results0[151] = {0};
        int results1[151] = {1};
        results1[N-1] = 1;
        int *results_final = cost0[N-1] < cost1[N-1] ? results0 :
results1;

        for(int i = N-1; i>0; i--){
            results0[i-1] = results0[i] ? back1[i] : back0[i];
            results1[i-1] = results1[i] ? back1[i] : back0[i];
        }

        for(int i = 0; i < N; i++){
            cout << results_final[i];
        }
        cout<<endl;
    }
}
```

Bài 5: Losing to Little Brother (Hard)

https://codejam.lge.com/contest/problem/648/5

Alice là chị gái của Albert. Hai chị em chơi trò chơi và vì Alice là chị gái của Albert nên Alice muốn nhường Albert. Alice sẽ chơi làm sao để thua Albert với số điểm nhỏ nhất.

Trò chơi như sau:

Trò chơi mà hai đứa trẻ này chơi sử dụng n thẻ, mỗi thẻ có một chữ cái tiếng Anh viết hoa (A-Z) trên đó.
Họ chơi trò chơi theo các quy tắc sau, và Alice biết chính xác Albert sẽ chơi như thế nào (vì anh ấy còn quá nhỏ!).

- Đầu tiên, hai bạn nhỏ sẽ xếp n thẻ xuống sàn từ trái sang phải theo thứ tự tùy ý.
- Lần lượt họ lấy một thẻ và người chơi chỉ có thể lấy thẻ ngoài cùng bên trái hoặc thẻ ngoài cùng bên phải.
- Albert đi trước, và sau đó họ thay phiên nhau.
- Trong lượt của Albert, nếu Albert lấy bất kỳ thẻ nào có một trong sáu chữ cái trong "ALBERT", thì Albert được 2 điểm.
- Trong lượt của Alice, nếu Alice lấy bất kỳ thẻ nào có một trong năm chữ cái trong "ALICE", thì Alice ghi được 1 điểm.
- Albert luôn lấy một thẻ theo mẫu sau:
- Nếu cả thẻ ngoài cùng bên trái và thẻ ngoài cùng bên phải đều có thể cho anh ta điểm, Albert luôn lấy thẻ ngoài cùng bên trái.
- Nếu không có thẻ nào trong hai thẻ có thể cho anh ta điểm, Albert luôn lấy thẻ ngoài cùng bên phải.
- Nếu chỉ một trong hai thẻ có thể cho anh ta điểm, Albert luôn lấy thẻ có thể cho anh ta điểm.

Alice muốn thua Albert, nhưng cô ấy muốn giảm thiểu sự khác biệt về điểm số, (Albert's score - Alice's score).

Ví dụ, giả sử rằng n = 4 và các thẻ trên sàn là "BCCB".

- Lượt 1: Albert sẽ lấy thẻ ngoài cùng bên trái "B", và ghi được 2 điểm. Các thẻ còn lại là "CCB".
- Lượt 2: Alice có thể lấy chữ "C" ngoài cùng bên trái hoặc chữ "B" ngoài cùng bên phải.
- Nếu cô ấy lấy chữ "C" ngoài cùng bên trái: Cô ấy ghi được 1 điểm, và "CB" vẫn còn. Albert lấy chữ "B" ngoài cùng bên phải và Alice lấy chữ "C" cuối cùng. Điểm của Albert là 4 và điểm của Alice là 2 (hiệu số là 2).
- Nếu cô ấy lấy chữ "B" ngoài cùng bên phải: Cô ấy không ghi được điểm nào và "CC" vẫn còn. Albert lấy chữ "C" ngoài cùng bên phải, và Alice lấy chữ "C" cuối cùng. Điểm của Albert là 2 và điểm của Alice là 1 (hiệu số là 1).
- Trong ví dụ này, nếu Alice cố gắng hết sức để thua Albert với cách biệt điểm nhỏ nhất, thì cô ấy chỉ có thể thua với cách biệt 1 điểm.

Trong một ví dụ khác, giả sử rằng n = 4 và các thẻ trên sàn là "CLCD".

- Lượt 1: Albert sẽ lấy thẻ ngoài cùng bên phải "D", và ghi 0 điểm. Các thẻ còn lại là "CLC".
- Lượt 2: Alice có thể lấy chữ "C" ngoài cùng bên trái hoặc chữ "C" ngoài cùng bên phải.
- Nếu cô ấy lấy chữ "C" ngoài cùng bên trái: Cô ấy ghi được 1 điểm và "LC" vẫn còn. Albert lấy chữ "L" ngoài cùng bên trái (và điểm 2) và Alice lấy chữ "C" cuối cùng. Đó là một trận hòa khi cả hai người chơi ghi được 2 điểm.
- Nếu cô ấy lấy chữ "C" ngoài cùng bên phải: Cô ấy ghi được 1 điểm, và "CL" vẫn còn. Albert lấy chữ "L" ngoài cùng bên phải (và điểm 2), và Alice lấy chữ "C" cuối cùng. Đó là một trận hòa khi cả hai người chơi ghi được 2 điểm.
- Trong ví dụ này, nếu Albert lần lượt lấy chữ "C" ngoài cùng bên trái, thì Alice có thể lấy chữ "D" ngoài cùng bên phải, tiếp theo là Albert lấy chữ "D" ngoài cùng bên phải và Alice lấy chữ "L" cuối cùng, sẽ có dẫn đến việc Alice thua 1 điểm. Tuy nhiên, Albert luôn lấy thẻ ngoài cùng bên phải khi không có thẻ nào cho anh ta điểm, và do đó Alice không thể thua Albert trong ví dụ này.

Đưa ra n thẻ trên sàn (từ trái sang phải), tính chênh lệch điểm số nhỏ nhất mà Alice có thể thua Albert.

1. Bruteforce

1.1. Nhận xét

- Albert luôn chơi theo quy luật cố định và biết trước -> Chỉ có thể lựa chọn ở lượt đi của Alice.
- Với mỗi lượt đi của mình, Alice có thể chọn quân bài ngoài cùng bên trái hoặc ngoài cùng bên phải -> O(2^(N/2)).
- Với việc sử dụng bruteforce, ta kiểm tra tất cả các lựa chọn của Alice, với mỗi lựa chọn, ta kiểm tra xem sau khi chơi xong, chênh lệch điểm giữa Alice và Albert là bao nhiêu. Kết quả của bài toán là số điểm chênh lệch nhỏ nhất.

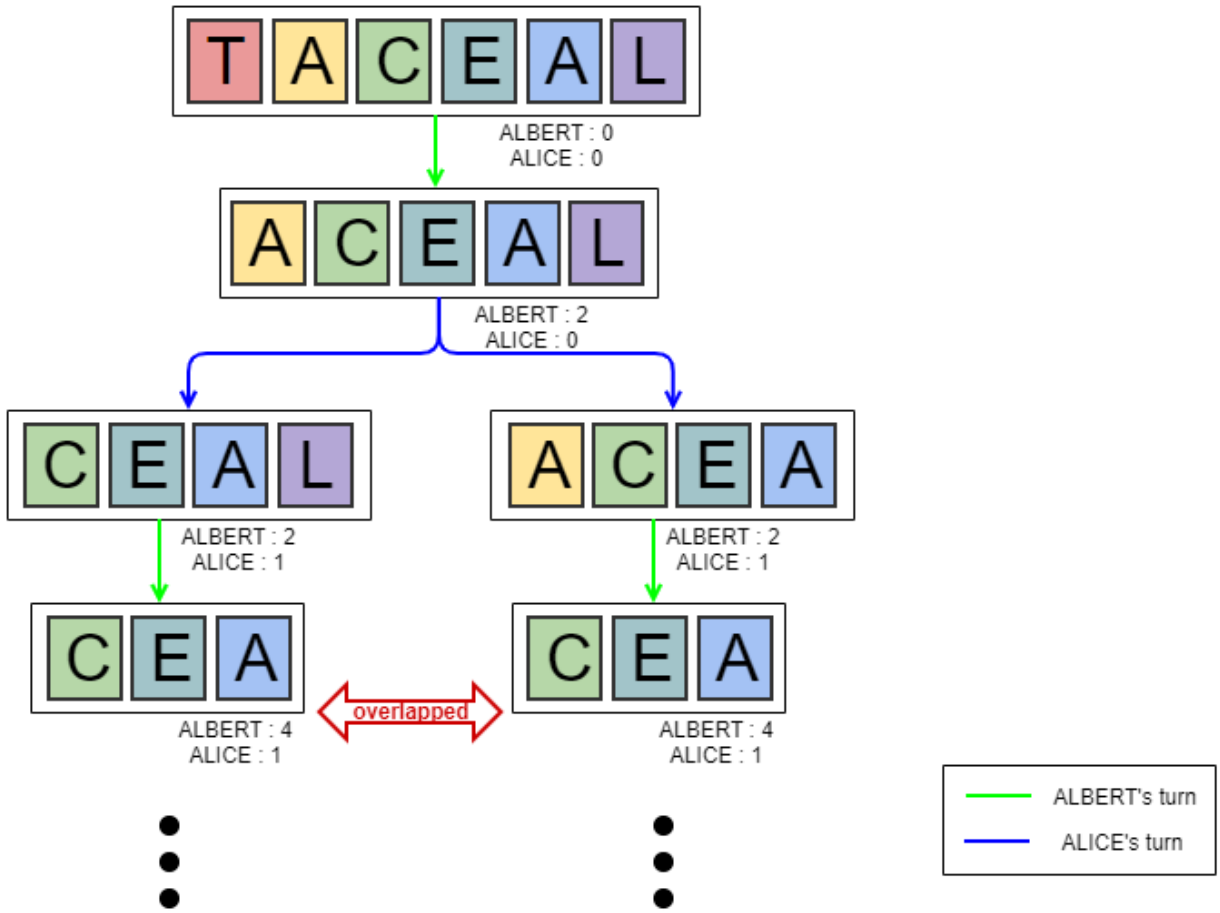
1.2. Giả code

```
void backtrack(int l, int r, int diff, bool isAlbert){
        if(l>r){
                if(diff>0){
                        ans=min(ans,diff);
                }
                return;
        }
        if(isAlbert){
                diff+=albertScore(l,r);
                backtrack(l,r,diff,false);
        } else {
                backtrack(l+1,r, diff + aliceScore(l+1,r),true);
                backtrack(l,r-1, diff + aliceScore(l,r-1),true);
        }
}
```

2. DP

2.1. Nhận xét

- Dễ dàng nhận thấy thuật toán bruteforce với N ~ 150 không đáp ứng được yêu cầu về thời gian.
- Dưới đây là mô tả các lựa chọn và điểm số tương ứng của Alice và Albert.



- Ban đầu: L=0, R=6, diff = 0
- Lượt đi 1 (Albert): L=1,R=6,diff = 2.
- Lượt đi 2 (Alice):

L=2, R=6, diff = 1	L=1,R=5, diff=1

- Lượt đi 3 (Albert):

L=2, R=5, diff = 3	L=2,R=5, diff=3

- Lượt đi 4 (Alice)

L=3,R=5,diff=2	L=2,R=4,diff=2	L=3,R=5,diff=2	L=2,R=4,diff=2

- Ta nhận thấy, ở lượt đi thứ 3, các vị trí của bộ bài còn lại (L và R) và điểm số chênh lệch của nhánh trái và nhánh phải (diff) là giống hệt nhau => Với 2 trường hợp này, ta chỉ cần kiểm tra một trường hợp, trường hợp còn lại sẽ có các cách lựa chọn và điểm số giống hệt (Mô tả ở lượt đi 4). Do ta chỉ cần quan tâm tới điểm số chênh lệch cuối cùng sau khi đã lấy hết bài trên sàn, nên với 2 trường hợp bị trùng lặp trong lượt 3, ta chỉ cần xem xét một trường hợp và bỏ qua trường hợp còn lại.

⇨ Ta có thể dùng một bảng nhớ để đánh dấu những trạng thái đã được xem xét (L,R,diff) để tránh phải kiểm tra nhiều trạng thái giống nhau => Loại bỏ những trường hợp trùng lặp.

2.2. Giả code

D(left, right, Gdiff) =

In ALBERT's turn,

(left-1)-th letter is in "ALBERT" → D(left-1, right, Gdiff-2)

(left-1)-th letter is NOT in "ALBERT",

(right+1)-th letter is in "ALBERT" → D(left, right+1, Gdiff-2)

(right+1)-th letter is NOT in "ALBERT" → D(left, right+1, Gdiff)

In ALICE's turn(OR both 1 & 2 conditions),

1. (left-1)-th letter is in "ALICE" → D(left-1, right, Gdiff+1)

(left-1)-th letter is NOT in "ALICE" → D(left-1, right, Gdiff)

2. (right+1)-th letter in "ALICE" → D(left, right+1, Gdiff+1)

(right+1)-th letter is NOT in "ALICE" → D(left, right+1, Gdiff)

2.3. Cài đặt thuật toán

```
void recurse(string const& S, int low, int high, int diff, bool turnAlbert, int&
ans) {
    if (mem[low][high][diff]) return;
    int curDiff = diff;
    if (low == high) {
        if (diff>0) {
            ans = min(ans, diff);
        }
        mem[low][high][diff] = true;
        return;
    }
    if (turnAlbert) {
        if (albert.find(S[low]) != std::string::npos) {
            recurse(S, low + 1, high, diff + 2, false, ans);
        } else {
            curDiff = diff;
            if (albert.find(S[high-1]) != std::string::npos) curDiff += 2;
            recurse(S, low, high - 1, curDiff, false, ans);
        }
    } else {
        curDiff = diff;
        if (alice.find(S[low]) != std::string::npos) curDiff -= 1;
        recurse(S, low + 1, high, curDiff, true, ans);

        curDiff = diff;
        if (alice.find(S[high-1]) != std::string::npos) curDiff -= 1;
        recurse(S, low, high - 1, curDiff, true, ans);
    }
    mem[low][high][diff] = true;
}
```