Two Pointers

Two pointers é uma tecnica muito interessante quando estamos trabalhando com dois indices. Temos milhares de aplicações de two pointer e hoje vou mostrar duas delas pra vocês, sendo que é bem facil utilizar essa ideia em outros problemas.

O primeiro problema é o seguinte:

Tenho uma array ordenada e que achar um par v[i] e v[j] tq v[i]+v[j] = x. Como Faço? (pedir pra pensarem)

algoritmo bobo pra fazer isso é:

```
bool temPar(A[], N, X){
   for (i = 0; i < N; i++) {
       for (j = 0; j < N; j++) {
           if (A[i] + A[j] == X  return true;
       }
   }
   return false;
}
```

Testo todos os pares, isso demora o O(n²).

Bom, usando agora a técnica de 2 ponteiros fazemos o seguinte:
setamos um i, no início da array;
um j no fim da array.
caso v[i] + v[j] < x , incrementamos o i;
caso v[i] + v[j] > x, decremento j;
até chegarmos na soma.

assim temos o seguinte código:

```
bool par(vector<int> v, x)
    int i = 0;
    int j = v.size() - 1;

    while (i < j) {
        if (A[i] + A[j] == X) return true;
        else if (A[i] + A[j] < X)  i++;
        else   j--;
    }
    return false
```

Isso tem complexidade O(n).
Resolução Vasya and Strings, mostrar o problema

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

int n, k;
string s;

int check(char x) {

	// Valor de retorno, ponteiro left e right
	int ret = 0, l, r = 0;
	// Quantos caracteres 'x' ja passei por
	int cnt = 0;
	// Pointer da esquerda esta dentro da string
	for (l = 0; l < n; l++) {

		while (r < n && (cnt < k || s[r] != x)) {

			if (s[r] == x)
				cnt++;
			r++;
		}

		if (s[l] == x)
			cnt--;
		if (ret < r-l)
			ret = r-l;
	}
	return ret;
}

int main(){

	cin >> n >> k;
	cin >> s;

	int ansa = check('a');
	int ansb = check('b');
	int ans;

	if (ansa > ansb)
		ans = ansa;

	else
		ans = ansb;

	cout << ans << endl;
}
```

Recursão

Definiremos recursão como uma função que chama ela mesma. De tal forma, podemos resolver varios problemas de um jeito simples e facil. Exemplos que podemos resolver usando essa tecnica: fibonacci, algoritimos em grafos (vao ver em aulas futuras) e o classico problema da torre de hanoi, que o bento vai explicar no fim da aula.

Bom, vamos mostrar um exemplo:

```
void prt(int n){
        cout << “MaratonUSP ” << n << endl;
        prt(n+1)
}
```

Mas porque usar recursão? Em muitas aplicações, implementar um codigo recursivo é muito mais facil doq implementar um codigo iterativo, geralmente, coisas que envolvem hierarquia e dependencia.

Ex factorial:

Como exemplo, vamos calcular o fatorial de um numero:
Fazer codigo iterativo e depois mostrar o codigo recursivo.

```
int fact(int n){
    if (n < = 1)
      return 1;
    else
      return n*fact(n-1);
```

Assim, a ideia é resolver o problema a cada passo até chegar no fim. Nesse codigo, vimos que a linha mais essencial é a primeira, chamada de base da recursão, que define o ponto de parada da execução.

Perguntar se eles conhecem algum outro tipo de problema em que poderiamos usar recursão.