Bitmask

- Operations:

__builtin_popcount(int mask) = retorna a quantidade de bits setados em "mask"
(valor << quantidade) = shift left
(valor >> quantidade) = shift right
(valor1 & valor2) = and bit-a-bit
(valor1 | valor2) = or bit-a-bit
(valor1 ^ valor2) = xor bit-a-bit
valor = ~valor; not

Para valores maiores que 109, usem (ll(valor) << quantidade)

- Box stacking:

```
typedef struct{
	int x;
	int y;
	int z;
}Box;
vector<Box> g;

int dp[13][3][(1<<11) + 5]; // lembre de resetar a dp toda vez que for usar

int solve(int last, int op, int bmask, int n){
	if(bmask == 0) return 0; // acabaram as caixas livres
	if(dp[last][op][bmask] != -1) return dp[last][op][bmask]; // esse estado já foi calculado anteriormente

	int ret = 0;

	for(int i=0;i<n;i++){
		if( (1 << i) & bmask){ // se o resultado for != 0, entao a caixa pode ser usada
			bmask = bmask^(1<<i); // seta a caixa como ocupada
			for(int j=0;j<3;j++){
				bool ok = valid(last,op,i,j); // verifica se eh possivel colocar a caixa i em cima da ultima caixa
				if(ok) ret = max(ret, 1 + solve(i,j,bmask,n)); // chama a recursao para uma nova combinacao
			}
			bmask = bmask|(1 << i); // a caixa que já foi usada, volta a ser livre
		}
	}
	return dp[last][op][bmask] = ret; // atualiza a dp com o valor calculado desse estado
}

int main( ){
	int n;
	cin >> n; // lê a entrada e adiciona no vector (... resto do código)
	int bmask = 0;
	for(int i=0;i<n;i++)
		bmask = bmask|(1<<i);// seta todos os bits (caixas) como livre.
	cout << solve(-1,0,bmask,n) << endl;
}
```