PROBLEMA DIFÍCIL 1

1 Análise Inicial

Nesse problema, precisa-se analisar uma string recebida pelo usuário para encontrar os padrões de diamantes, dados por ”<>”. Então, para resolver isso, decidiu-se contar a quantidade de caracteres ”<”que aparecerem e usar esse valor como flag para contar quantos diamantes podem ser formados. Assim, sempre que encontrar um caractere ”>”e o flag tiver um valor maior que zero, pode-se contar um diamante e reduzir um do valor da flag. Com isso, sempre que aparecer um ”<”e um ”>”um diamante será contado.

2 Resolução do Problema

Primeiramente, foram criadas as variáveis que seriam usadas, começando por um inteiro ”N”que recebe a quantidade de casos que serão analisados e o vetor ”string”que receberá os casos. O vetor é criado com 1001 espaços pois a quantidade máxima de caracterees que devem ser lidos é de 1000, mais um para o espaço de NULL criado pela função fgets. Então, lê-se a quantidade de casos e aloca-se o espaço do vetor de resultados.

```
int main(void) {
  int counter = 0, flag = 0, N = 0, *results = NULL;
  char string[1000 + 1]; //1000 caracteres de <,> e . mais 1 para guardar o
NULL do fgets

  printf("Digite o numero de casos:\n");
  scanf("%d", &N);
  while(getchar() != '\n' && getchar() != EOF); //Limpa o buffer

  results = malloc(N*sizeof(int)); //Aloca a quantidade necessária para a
lista de resultados
```

Figura 1: Criação das Variáveis e Alocação Dinâmica do Vetor de Resultados

Com isso, fez-se um laço para analisar cada caso dado. Primeiramente, zerou-se a variável de flag e o contador (”counter”) e leu-se a string enviada pelo usuário. Com isso, fez-se outro laço para percorrer a string com a variável ”pos”até que fosse encontrado o caractere ”\0”, que marca o final da string.

Nesse laço, verifica-se cada caractere dentro da string e o compara com ”<”, caso seja encontrada uma igualdade, adiciona-se 1 na flag, isso quer dizer que é possível formar 1 diamante com o caractere ”<”. Além disso, também verificam-se os caracteres ”>”e, caso a flag esteja levantada, conta-se um diamante e reduz 1 da flag, que significa que um dos diamantes que podiam ser formados já foi retirado. Dessa maneira, é possível contar todos os diamantes possíveis de ser formados sem precisar alterar a string em si.

Ao final de cada string, a contagem de diamantes é armazenada no vetor de resultados, então, as variáveis de flag e contador são zeradas e analisa-se um novo caso.

```
  for(int i = 0; i < N; i++){
    counter = 0;
    flag = 0;

    printf("\nDigite o caso de teste %d:\n", i+1);
    fgets(string, sizeof(string), stdin); //Recebe as strings de diamantes

    for(int pos = 0; string[pos] != '\0' && string[pos] != EOF; pos++){
      if(string[pos] == '<'){ //Se encontrar um "inicio de diamante"
        flag++; //Aumenta 1 na flag para contar quantos diamantes podem ser
formados
      }
      else if(string[pos] == '>' && flag != 0){ //Se um diamante pode ser
formado
        counter++; //Conta um diamante formado
        flag--; //Tira um da quantidade de diamantes que podem ser formados
      }
    }

    results[i] = counter; //Guarda um resultado na lista
  }
```

Figura 2: Laços de Contagem de Cada Caso

Com isso, ao terminar o laço principal, obtem-se um vetor de resultados com os valores dos números de possíveis diamantes de cada caso apresentado, então, apenas imprimem-se os resultados na tela e libera-se o espaço alocado dinamicamente.

```
    for (int i = 0; i < N; i++) printf("%d\n", results[i]);

    free(results);

    return 0;
}
```

Figura 3: Resultados Finais e Liberação da Memória