Podczas laboratorium Kontynuujemy pracę z algorytmami dotyczącymi operacji na ciągach znaków, a konkretnie z kompresją.

Pamiętajcie, że macie to zrealizować algorytmicznie (korzystając z pętli, instrukcji sterujących, etc), a nie „językowo" (użycie sorter() czy find()).

1. **Kodowanie długości serii – proste.**
   Algorytm powinien działać następująco. Wczytany tekst zamienia ciąg liczb odpowiadających kodom ASCII każdego ze znaków. Następnie, każda z liczb zamienia na wartość 8bitową binarną i zapisywana do ciągu. Np. dla ciągu znaków „!AB*" otrzymamy 00100001 01000001 01000010 00101010. W kolejnym kroku zapisujemy informację o kolejnych długościach ciągów na przemian 0 i jedynek. Czyli dla powyższego przykładu uzyskujemy 2 1 4 1 1 1 5 1 1 1 4 1 3 1 1 1 1 1 1 (jeśli czegoś nie pomyliłem 😉).

   Następnie należy kolejne wartości z ciągu 2 1 4 … zamienić na znaki (*char*) i wstawić do ciągu wyjściowego (dla małych wartości będą to znaki niewidoczne – patrz tabela ASCII w sieci). Jest tu mały haczyk, otóż teoretycznie można trafić na ciąg 0 lub 1 który przekracza wartość 255 (8bitowe kody ASCII mają 256 wartości od 0 do 255). Co wtedy? Trzeba taką wartość podzielić na kilka. Rozważmy na przykładzie (kolejność 0 i 1 nie istotna, ważne są liczby): 500 20 10 700 25. Należy w takim wypadku 500 podzielić na 255 i 245, ale nie mogą one nastąpić po sobie (inaczej będziemy mieli np. 255 jedynek i 245 zer), więc uzyskamy coś takiego: 255 0 245 potem 20 10 bez zmian i znów mamy coś większego od 255. Dzielimy 700 na 255 + 255 +190, a nasz kompletny zapis ciągu 500 20 10 700 25 wyglądałby następująco w wersji podzielonej:
   255 0 245 20 10 255 0 255 0 190 25.

2. **Kodowanie długości serii – rozbudowane.**
   Wykonujemy zadanie jak wyżej, ale liczba bitów na których zapisujemy długość ciągu jest zmienna i wynika bezpośrednio z największej długości ciągu. Np. dla powyższego 500 20 10 700 liczba bitów wynikałaby z wartości 700. Na 8 bitach możemy zapisać maksymalnie wartość 255, na 9 bitach wartość 511, na 10 bitach wartości od 0 do 1023. Czyli dla tego przypadku użylibyśmy 10 bitów.

Pozostają dwie nierozwiązane kwestie:

1. Skąd przy dekodowaniu użytkownik ma wiedzieć że to zapis po 10 bitów a nie np. po 3 bity?
2. Co w wypadku gdy mamy 5 wartości, każdą zapisaną na 10 bitach? Musielibyśmy zapisać za mało (6*8 = 48) lub za dużo (7*8 = 56) Bajtów informacji.

Rozwiązanie 1. punktu jest dość proste. Dla uproszczenia zakładamy że maksymalna długość ciągu nie może przekraczać 256, więc możemy zapisać w pierwszym znaku (*char* ma 8 bitów czyli wartości od 0 do 255 możemy tam przechować, jeżeli przyjmiemy za podstawę długość 1 to przechowamy wartości od 1 do 256) informację o liczbie bitów wykorzystanej do zapisania długości.

Punkt 2 możemy również w miarę prosto rozwiązać. Zapisujemy nadmiarowe bity. Kwestia tego ile tych bitów? Od 1 (gdy musimy dopełnić z 7 do 8) do 7 włącznie (gdy dopełniamy z 1 do 8 bitów). Żeby przechować informację z zakresu od 0 (gdy nic nie dopełniamy, bo mamy wielokrotność 8) do 7 wystarczą nam trzy bity (od $000_2 = 0_{10}$ do $111_2 = 7_{10}$). Więc mielibyśmy 8 bitów na informację o długości N, potem 3 bity informacji o liczbie R nadmiarowych i *liczba_ciągów* * N bitów samych danych, po których nastąpiłoby od 0 do 7 bitów nadmiarowych. Przez nawias klamrowy będę oznaczać poniżej jeden Bajt informacji. Mniej więcej tak zapisalibyśmy kolejne Bajty skompresowane powyższą metodą: [N – liczba bitów][R + 5 bitów danych] [][][]…[][ostatnie bity danych + 0-7 nadmiarowych]