algorytm, umożliwiający obliczenie dla każdej pozycji i maksymalnego promienia $R[i]$ palindromu o środku na pozycji i (promieniem palindromu vv^R jest długość v), a dokładniej

$$R[i] = \max\{k \geq 0: x[i-k+1..i] = (x[i+1..i+k])^R\}$$

```
{Algorytm Manachera; obliczanie promieni słów symetrycznych}
begin
    {przyjmijmy, że x[1] = $, x[n] = #}
    R[1] := 0; i := 2; j := 0;
    while (i ≤ n) do
    begin
        while x[i − j] = x[i + j + 1] do j := j + 1;
        R[i] := j;
        k := 1;
        while (R[i − k] ≠ R[i] − k) and (k ≤ j) do
        begin R[i + k] := min(R[i − k], R[i] − k); k := k + 1 end;
        j := max(j − k, 0); i := i + k
    end
end {algorytm Manachera};
```

Poprawność algorytmu wynika z następującej własności promieni palindromów: jeśli dla $k = 1..R[i]$ mamy $R[i-k] \neq R[i]-k$, to $R[i+k] = \min(R[i-k], R[i]-k)$.

Operacjami dominującymi w algorytmie są porównania $x[i-j] = x[i+j+1]$. Porównań takich z wynikiem negatywnym jest wykonywanych co najwyżej n, a dla każdej wartości i co najwyżej jedno. W porównaniach z wynikiem pozytywnym wzrasta wartość sumy $i+j$. Wartość ta (przy porównywaniu symboli) nie maleje. Ponieważ maksymalną wartością $i+j$ jest n, całkowita liczba wykonanych porównań symboli nie przekracza $2n$.

5.3.5. Równoważność cykliczna

Dwa słowa są cyklicznie równoważne, gdy są równe w sensie list cyklicznych, co zapisujemy jako $x \equiv y$. Problem polega na sprawdzeniu, czy dwa dane słowa są cyklicznie równoważne.

Wystarczy skorzystać z następującego faktu: $x \equiv y$ wtedy i tylko wtedy, gdy x występuje (jako wzorzec) w tekście yy. Zakładamy tutaj, że $|x| = |y|$. Problem sprowadza się więc do problemu WW i jego koszt jest liniowy. Istnieje jednak prostszy algorytm, który nie wymaga wyszukiwania wzorca i korzystania z dodatkowej tablicy (pamięć $O(1)$). Niech < będzie dowolnym porządkiem liniowym na zbiorze symboli.


```
{Algorytm sprawdzania, czy u ≡ w; niech x = ww$, y = uu#, n = |u|}
begin
    i := 0; j := 0; k := 1;
    while (i < n) and (j < n) and (k ≤ n) do
    begin
        k := 1;
        while x[i + k] = y[j + k] do k := k + 1;
        if (k ≤ n) then
        begin if x[i + k] > y[j + k] then i := i + k else j := j + k end;
        {niezmiennik *}
    end;
    if (k > n) then return (u ≡ w) else return (nie zachodzi u ≡ w)
end {algorytm};
```

Niech $u^{(k)} = u[k+1..n]u[1..k]$, a więc niech $u^{(k)}$ powstaje przez cykliczne przesunięcie u. Niech $D1 = \{1 \leq k \leq n : w^{(k-1)} \gg u^{(j)}$ dla pewnego $j\}$ i niech $D2 = \{1 \leq k \leq n : u^{(k-1)} \gg w^{(j)}$ dla pewnego $j\}$, gdzie $\gg$ oznacza rozszerzenie $>$ na słowa (leksykograficznie). Poprawność algorytmu wynika stąd, że jeśli $D1$ lub $D2$ jest zbiorem $\{1, 2, ..., n\}$, to nie zachodzi $u \equiv w$. Ponadto zachodzi niezmiennik $*$: $\{1, 2, ..., i\} \subseteq D1$, $\{1, 2, ..., j\} \subseteq D2$. Algorytm ma złożoność liniową. Największa liczba porównań symboli jest wykonywana dla słów u i w postaci (odpowiednio) 11...1201, 111...120.

5.3.6. Algorytm Huffmana

Kompresję tekstów można rozumieć na różne sposoby. Dla nas będzie ona oznaczać redukcję binarnego zapisu danego tekstu. Załóżmy, że dla danego tekstu $x = a_1a_2...a_n$ i alfabetu I chcemy znaleźć jednoznaczny kod h symboli a_i alfabetu słowami binarnymi $h(a_i)$ tak, żeby długość tekstu $h(a_1) \cdot ... \cdot h(a_n)$ była minimalna. Rozważamy tutaj klasę takich kodowań, że $h(a_i)$ nie jest prefiksem $h(a_j)$ dla żadnych dwóch różnych symboli a_i i a_j. Zbiór kodów symboli możemy reprezentować drzewem, którego ścieżki odpowiadają kodom poszczególnych symboli. Bit 0 oznacza „idź w lewo" w drzewie, a bit 1 – „idź w prawo".

Rozważmy przykład $x = abcdcdcddb$. Jeżeli zakodujemy każdy symbol ciągiem dwubitowym, to otrzymany kod będzie miał długość $2n = 20$. Przedstawimy teraz algorytm opracowany przez D. Huffmana. Istota tego algorytmu sprowadza się do kodowania częściej występujących symboli krótszymi ciągami binarnymi, a rzadziej występujących symboli – dłuższymi. Przedstawimy na naszym przykładzie działanie algorytmu w wersji rekurencyjnej. Częstość występowania w słowie x poszczególnych symboli jest następująca:

$$częstość(a) = 1,\ częstość(b) = 2,\ częstość(c) = 3,\ częstość(d) = 4.$$