Algorytmy i programowanie/ Algorytmy i struktury danych

Algorytm pisania pracy na studiach...

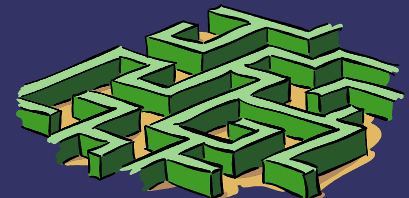
PISANIE PRACY?
Napisałeś już tytuł?
TO NAPISZ!
Masz już napisaną?
IDŹ DO LEKARZA!
Masz już Bibliografię?
Sprawdzałeś już maila?
Sprawdzałeś już FB?
Wysyłałeś już SMSy do innych: "Jak idzie?"
Sąsiad robi remont?
Głód?
Pies musi na spacer?
Piwo?
Dużą?
SPOKO :)
I TAK CI ONA NIC NIE DA
Jedzenie?
Odchudzasz się?
TO IDŹ POĆWICZYĆ
Drzemka?
Długa?
Idziesz spać?
JUTRO ZACZNIESZ
PRZERWA?
Zrobiło się późno?
Sprawdzanie czegoś na wikipedii?
Atak terrorystyczny w pobliżu?
Jesteś pewien?
Dokarmianie gołębi?
Pomoc charytatywna?

by Emily

celebixy.pl F*** Me I'm Famous

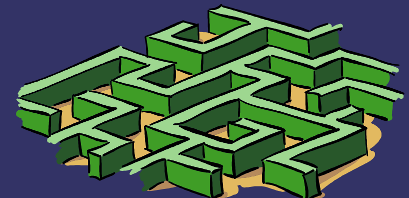
Plan prezentacji

- Definicja algorytmu
- Przykłady algorytmów
- Metody rozwiązywania problemów
- Pseudokody algorytmów:
- - wyszukiwania,
- - sortowania,
- -
- Struktury danych:
- - stosy,
- - kolejki,
-



Definicja algorytmu

- Algorytm to skończony ciąg jasno zdefiniowanych czynności koniecznych do wykonania pewnego rodzaju zadania w skończony czasie.
-
- Klasycznym przykładem algorytmów są przepisy kulinarne, gdzie trzeba postępować według określonych czynności.



Przykłady algorytmów

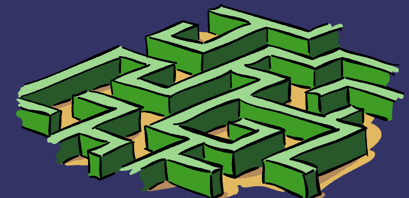
- Algorytmy kompresji - zmiana zapisu informacji, tak aby zmniejszyć nadmiarowość i tym samym objętość zbioru. Zadaniem kompresji jest wyrażenie tego samego tylko za pomocą mniejszej liczby bajtów
- Kryptologia – zabezpieczenie informacji przed niepowołanym dostępem.

Metoda „dziel i zwyciężaj”

Problem dzieli się rekurencyjnie na dwa lub więcej mniejszych podproblemów tego samego lub podobnego typu tak długo aż staną się wystarczająco proste do bezpośredniego rozwiązania. Następnie całość się scala, by podać rozwiązanie całego problemu. Przykładami algorytmów wykorzystujących tę technikę są: wyszukiwanie binarne, quicksort i mergesort.

Algorytmy wyszukiwania – wyszukiwanie liniowe

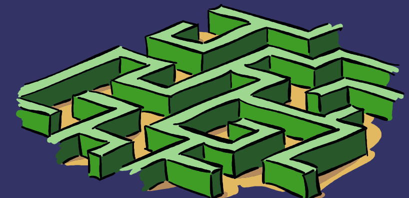
- Algorytm wyszukiwania liniowego ma złożoność O(n), ponieważ musi przeglądnąć wszystkie elementy tablicy.
- znajdzIndeksElementuLiniowo(A, n, x):
-     i = 1;
-     while (A[i] != x && i != n):
-         i++;
-     if (A[i] !=x):
-         return false;
-     else:
-         return i;



Wyszukiwanie binarne - algorytm

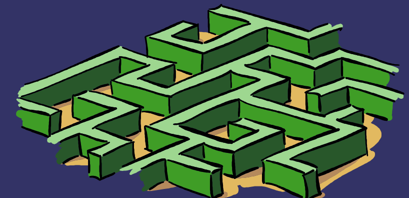
Ten algorytm zakłada, że tablica jest posortowana

```
znajdzIndeksElementu(A, n, x):
    i = 1;
    j = n;
    k = n/2;
    while (A[k] != x || i < j):
        k = (i+j)/2;
        If (x > A[k]):
            i = k+1;
        Else:
            J = k-1;
    If (A[k] == x):
        return k;
    return false;
```



Wyszukiwanie binarne – specyfikacja wejścia i wyjścia

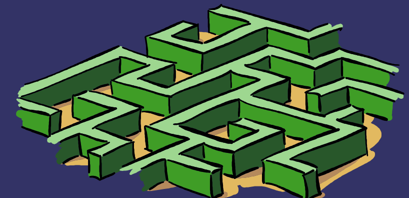
- Poprzedni algorytm przyjmuje:
- A – tablica obiektów pewnego typu,
- n – rozmiar tablicy
- x – szukany element w tablicy
- Tablica posiada własność:
- A[1] < A[2] < … < A[n-1] < A[n]
-
- Złożoność algorytmu : O(log2 n)



Algorytmy sortowania: sortowanie przez scalanie

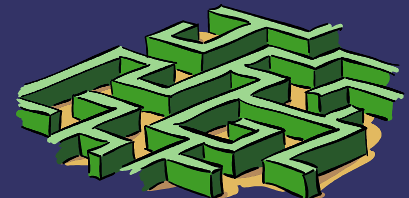
- Algorytm wykorzystuje metodę dziel i zwyciężaj.
-

```
SORTUJ_I_SCAL(A, p, r):
    if (p < r):
        q = p+r/2;
        SORTUJ_I_SCAL(A, p, q);
        SORTUJ_I_SCAL(A, q+1, r);
        SCAL(A, p, q, r);
```



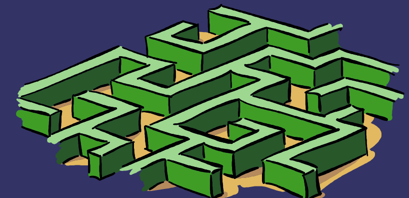
Algorytmy sortowania – scalanie posortowanych ciągów

```
SCAL(A, p, q, r):
    i = p; j = q+1; k = p;
    while (i <= q && j <= r):
        if (A[i] < A[j]):
            B[l] = A[i]; i ++;
        else:
            B[l] = A[j]; j++;
        l++;
    while (i <= q):
        B[l] = A[i]; i++; l++;
    while (j <= r):
        B[l] = A[j]; j++; l++;
    for (i = p to r):
        A[i] = B[i];
```



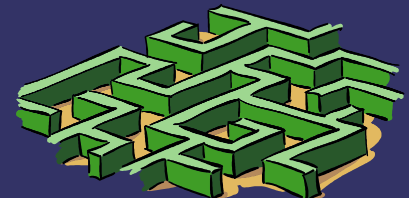
Algorytmy sortowania – sortowanie przez wstawianie

- Mając posortowaną listę elementów A[1..j-1] wstawiamy element A[j] we właściwe miejsce w tablicy, otrzymując większą posortowaną listę.
- SORTUJ_WSTAW(A, n):
-     for (j=2 to n):
-         r = A[j];
-         i = j-1;
-         while (i > 0 && A[i] > r):
-             A[i+1] = A[i]; i++;
-         A[i+1]=r;



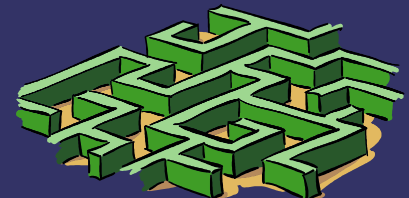
Algorytmy sortowania – sortowanie przez zliczanie

- Założenie: mamy ciąg n elementów będących liczbami całkowitymi z przedziału od 0 do ustalonego k. Lista kroków:
- Wyznacz liczbę elementów tablicy A równych i (i=0, .., k).
- Wyznacz liczbę elementów tablicy A równych lub mniejszych od i.
- Umieść wszystkie elementy na właściwych pozycjach tablicy B.



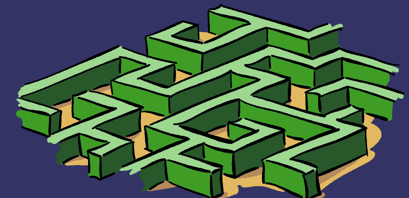
Algorytmy sortowania – sortowanie przez zliczanie (PSEUDOKOD)

```
C=[];
for i=0 to k:
    C[A[i]] = C[A[i]]+1;
for i=1 to k:
    C[i] = C[i]+C[i-1];
B=[];
for i=n downto 1:
    B[C[A[i]]]=A[i];
    C[A[i]] = C[A[i]]-1;
```



Algorytmy szybkiego sortowania

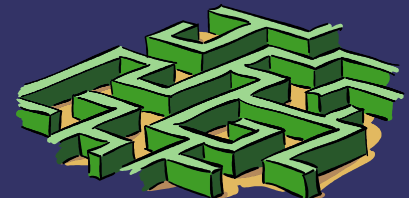
- Quicksort.
- Algorytm polega podzieleniu tablicy w miejscu x takiego, że wszystkie elementy z A[1..x-1] są mniejsze od A[x] natomiast elementy z tablicy A[x+1..n] (n- rozmiar tablicy) większe od A[x]. Algorytm wykorzystuje metodę „dziel i zwyciężaj”.



Algorytmy szybkiego sortowania

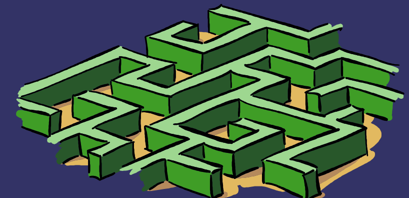
```
QUICKSORT(A, p, r): //dla całej tablicy QUICKSORT(A, 1, n).
    if (p < r):
        q = podzial(A, p, r);
        QUICKSORT(A, p, q-1);
        QUICKOSRT(A, q+1, r);

PODZIAL(A, p, r):
    x = A[r];
    i = p-1;
    for j=p to r-1:
        if (A[j] <= r):
            j++;
            swap(A[i], A[j]);
        swap(A[i+1], A[r]);
    return x;
```



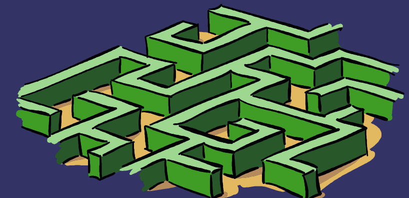
Algorytmy grafowe – przeszukiwanie w głąb i wszerz

- Przy przeszukiwaniu w głąb wykorzystujemy strukturę stosu zapamiętującą nam nieodwiedzone jeszcze następniki kolejnych wierzchołków zaczynając od tego, który jest najdalej.
- Przy przeszukiwaniu wszerz pomocną jest struktura kolejki, a przy przeszukiwaniu najpierw odwiedzamy sąsiadów kolejnych wierzchołków, które są najbliżej.



Stosy

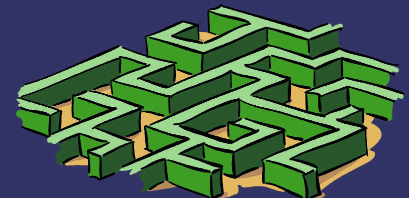
- Struktura danych implementującą zbiór dynamicznym o stałej złożoności operacji dodawania i usuwania elementu ze zbioru.
- W stosie panuje zasada LIFO (last in, first out) czyli pierwszy wychodzi, ten który ostatni przyszedł. Dotyczy to operacji dodawania i usuwania. Złożoności operacji PUSH i POP wynoszą O(1).



Stos – podstawowe operacje

- PUSH
  umieszcza element na stosie, rozmiar stosu zwiększa się o jeden.

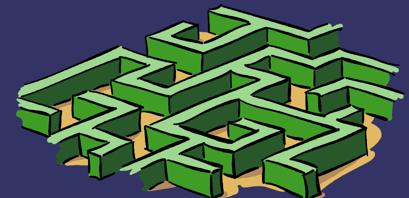
- POP
  usuwa element ze stosu i go zwraca, zmniejsza rozmiar stosu lub zwraca błąd w przypadku pustego stosu.



Stos – podstawowe operacje

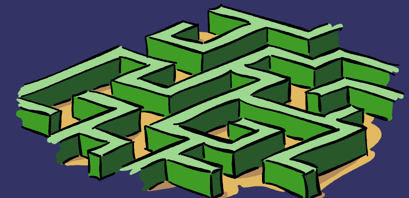
TOP

- Zwraca odwołania na elementu na wierzchu stosu bez usuwania go. Zwraca błąd przy pustym stosie.
- SIZE
  zwraca liczbę elementów na stosie
- EMPTY
  sprawdza czy stos jest pusty
- CLEAR
  czyści stos, stos staje się pustym.



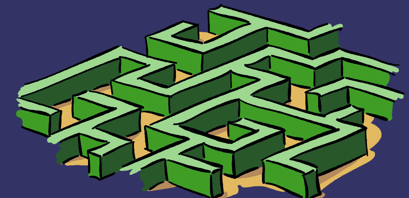
Stos – przykładowe implementacje

- Dane stosu będą przechowywane w tablicy S[1..n], zakładając, że elementów jest nie więcej niż n.
- Mamy do dyspozycji atrybut top(S) który posiada indeks ostatnio dodanego elementu
- PUSH(S, x)
-     top(S) = top(S)+1;
-     if (top(S) > n):
-         Error;
-     else
          S(top(S)) = x;



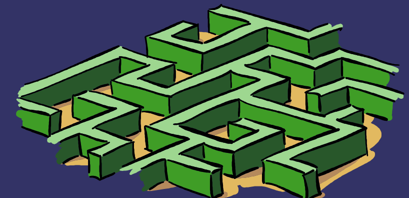
Operacje na stosie

- POP(S):
- if (EMPTY(S)):
-   error();
- else:
-   val =S(top(S));
-   top(S)=top(S)-1;
-   return val;
-



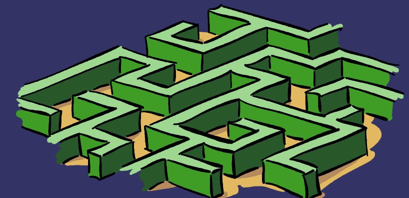
Kolejki

- Kolejka jest implementacją zbioru dynamicznego, w którym dodawany element znajdzie się na początku tablicy i usuwany element jest z końca tablicy. Jest typem zbioru dynamicznego FIFO (first in, first out).
- Operacje na kolejce to: enqueue, dequeue, i pozostałe (poza push i pop) dla stosu są takie same.



Kolejka cykliczna - implementacje

- Tablica Q[1..n] oraz atrybuty head(Q) i tail(Q) zawierające indeks pierwszego elementu oraz indeks pierwszej wolnej pozycji. Zakładamy, że indeks 1 jest następnikiem indeksu n. Początkowo head(Q) = tail(Q).
- ENQUEUE(Q, x):
-     Q[tail(Q)] = x;
-     If (tail(Q) == n):
-         Tail(Q) = 1;
-     Else tail(Q) = tail(Q)+1;



Kolejka – usuwanie elementu

- Dequeue(Q):
- x = Q[head(Q)];
- if (head(Q) == n):
  head(Q) = 1;
- Else:
- head(Q) = head(Q)+1;
- return x;