Sortowania

1. Quicksort :

* O(nlogn), najgorzej O(n^2)
* Niestabilny
* Najpopularniejszy

1. MergeSort:
   * O(nlogn), ale wolniejszy od QS
   * Stabilny
2. Heapsort:
   * O(nlogn)
   * Przez kopiec maksymalny (maxheap)
3. Insertionsort
   * O(n^2)
   * Stabilny
4. SelectionSort
   * O(n^2)
   * Niestabilny
5. RadixSort
   * Do sortowania np. tekstu
   * Jak spełnione warunki to O(n\*k) k-długość słowa
6. CountSort
   * O(n\*k),k-długość alfabetu
   * Liczy ilość znaków i potem je wypisuje w posortowanej kolejności
7. BucketSort
   * O(n\*k) k-to ilośc bucketów, na które dzielimy, warunkiem jest to, żeby był jednorodny rozkład

KomiWoj

**Rekurencyjna procedura TSP ( n, graf, v, v o, d, d H, S, S H, visited )**

**Wejście:**

|  |  |  |
| --- | --- | --- |
| *n* | – | liczba wierzchołków w grafie, *n*  ∈ C. |
| *graf* | – | zadany w dowolnie wybrany sposób, algorytm tego nie precyzuje. Definicja grafu powinna udostępniać wagi krawędzi. |
| *v* | – | wierzchołek bieżący, *v*  ∈ C. |
| *v 0* | – | wierzchołek startowy, *v 0*  ∈ C. |
| *d* | – | suma wag krawędzi cyklu Hamiltona, *d*  ∈ C. |
| *d H* | – | pomocnicza suma wag krawędzi, *d H*  ∈ C. |
| *S* | – | stos wierzchołków. |
| *S H* | – | pomocniczy stos wierzchołków. |
| *visited* | – | *n*  elementowa tablica logiczna odwiedzin. |

**Wyjście:**

|  |  |  |
| --- | --- | --- |
| *S* | – | stos zawierający numery kolejnych wierzchołków cyklu Hamiltona o najmniejszej sumie wag krawędzi lub stos pusty w przypadku braku cyklu Hamiltona w grafie. |
| *d* | – | suma wag krawędzi cyklu Hamiltona. |

**Zmienne pomocnicze:**

|  |  |  |
| --- | --- | --- |
| *u* | – | wierzchołek, *u*  ∈ C. |

|  |  |  |
| --- | --- | --- |
| K01: | *S H*.push ( *v*  ) | Odwiedzony wierzchołek dopisujemy do ścieżki |
| K02: | **Jeśli** *S H*  nie zawiera *n*  wierzchołków, **to idź do** kroku K10 | Jeśli brak ścieżki Hamiltona, przechodzimy do wyszukiwania |
| K03: | **Jeśli** nie istnieje krawędź z *v*  do *v 0*, **to idź do** kroku K17 | Jeśli ścieżka Hamiltona nie jest cyklem, odrzucamy ją |
| K04: | *d H*  ← *d H*  + waga krawędzi z *v*  do *v 0* | Uwzględniamy w sumie wagę ostatniej krawędzi cyklu |
| K05: | **Jeśli** *d H*  ≥ *d*, **to idź do** kroku K08 | Jeśli znaleziony cykl jest gorszy od bieżącego, odrzucamy go |
| K06: | *d*  ← *d H* | Zapamiętujemy sumę wag cyklu |
| K07: | Skopiujstos S H do stosu S | oraz sam cykl Hamiltona |
| K08: | *d H*  ← *d H*  - waga krawędzi z *v*  do *v 0* | Usuwamy wagę ostatniej krawędzi z sumy |
| K09: | **Idź do** kroku K17 |  |
| K10: | *visited* [ *v*  ] ← true | Wierzchołek zaznaczamy jako odwiedzony, aby nie był ponownie wybierany przez DFS |
| K11: | **Dla** każdego sąsiada *u*  wierzchołka *v* : **wykonuj** kroki K12...K15 | Przechodzimy przez listę sąsiedztwa |
| K12: | **Jeśli** *visited* [ *u*  ] = true,     **to następny obieg pętli** K11 | Omijamy wierzchołki odwiedzone |
| K13: | *d H*  ←  *d H* + waga krawędzi z *v*  do *u* | Obliczamy nową sumę wag krawędzi ścieżki |
| K14: | TSP ( *n, graf, u, v o, d, d H, S, S H, visited*  ) | Wywołujemy rekurencyjnie poszukiwanie cyklu |
| K15: | *d H*  ←  *d H*  - waga krawędzi z *v*  do *u* | Usuwamy wagę krawędzi z sumy |
| K16: | *visited* [ *v*  ] ← false | Zwalniamy bieżący wierzchołek |
| K17: | *S H*.pop( ) | Usuwamy bieżący wierzchołek ze ścieżki |
| K18: | **Zakończ** |  |

**Lista kroków algorytmu głównego**

|  |  |  |
| --- | --- | --- |
| K01: | Utwórz i wyzeruj *visited*, *S*  i *S H* |  |
| K02: | *d*  ← ∞;  *d H*  ← 0 | Początkową sumę ustawiamy jako największą z możliwych |
| K03: | TSP ( *v 0*  ) | Wyszukiwanie cyklu Hamiltona rozpoczynamy od wierzchołka startowego |
| K04: | **Jeśli** *S*.empty( ) = false, **to pisz** *S*  oraz *d* **inaczej pisz** "NO HAMILTONIAN CYCLE" | Sprawdzamy, czy algorytm znalazł cykl Hamiltona. Jeśli tak, to wypisujemy go. |
| K05: | **Zakończ** |  |

Czerwono Czarne

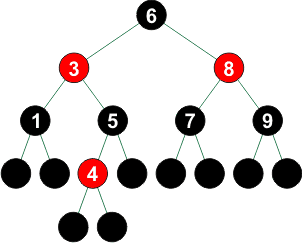
Operacja wstawiania węzła do drzewa czerwono-czarnego składa się z następujących etapów:

Tworzymy nowy węzeł, inicjujemy go danymi, po czym wstawiamy do drzewa czerwono-czarnego za pomocą zwykłej procedury wstawiania węzła do drzewa binarnego (ponieważ drzewo czerwono-czarne wciąż jest drzewem binarnym ). Gdy węzeł znajdzie się w drzewie, kolorujemy go na czerwono i sprawdzamy czy nie zostały zaburzone warunki drzewa czerwono-czarnego.



Jeśli nowy węzeł jest korzeniem, to zmieniamy jego kolor na czarny (ponieważ korzeń drzewa musi być czarny ). Wstawianie jest zakończone.

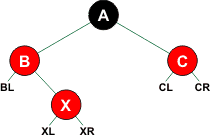
Jeśli ojciec nowego węzła jest czarny, to również kończymy, ponieważ warunki drzewa czerwono-czarnego w takim przypadku nie zostały zaburzone:



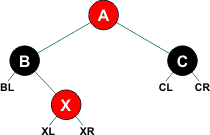
W przeciwnym razie mamy dwa kolejne węzły czerwone i musimy przywrócić warunki drzewa czerwono-czarnego (zauważ, że czerwony ojciec nowego węzła nie może być korzeniem z uwagi na warunek nr 3: korzeń drzewa musi być czarny ). W tym celu rozważamy 3 przypadki, które występują w dwóch lustrzanych postaciach (z zamienionymi stronami: lewa z prawą ):

**Przypadek 1:**

Wujek **C** (brat ojca **B,** czyli drugi syn węzła **A**) wstawionego węzła **X** jest czerwony.



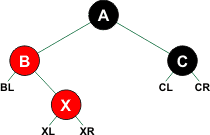
Ojca **B** i wujka **C** kolorujemy na czarno. Dziadka **A** kolorujemy na czerwono.



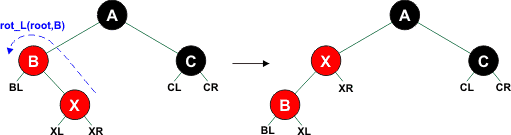
Jeśli węzeł **A** jest korzeniem drzewa, to zmieniamy jego kolor na czarny (ponieważ korzeń drzewa musi być czarny) i kończymy. W przeciwnym razie za nowe **X** przyjmujemy **A** i sprawdzamy od początku kolejne przypadki na wyższym poziomie drzewa.

**Przypadek 2:**

Wujek **C** jest czarny, a węzeł **X** jest prawym dzieckiem węzła **B** (przypadek lustrzany: wujek **B** jest czarny, a **X** jest lewym dzieckiem węzła **C** ).



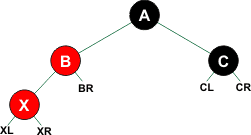
Wykonujemy rotację w lewo względem węzła **B** (w przypadku lustrzanym wykonujemy rotację w prawo względem **C** ).



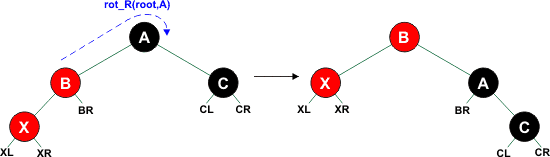
Za **X** przyjmujemy węzeł **B** i przechodzimy do przypadku 3.

**Przypadek 3:**

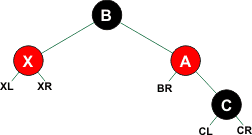
Wujek **C** jest czarny, a węzeł **X** jest lewym dzieckiem węzła **B** (przypadek lustrzany: wujek **B** jest czarny, a **X j**est prawym dzieckiem węzła **C** ):



Wykonujemy rotację w prawo względem węzła **A** (w przypadku lustrzanym wykonujemy rotację w lewo względem **A** ):



Zmieniamy kolory węzłów **A** i **B** na przeciwne:



Operacja wstawiania zostaje zakończona, ponieważ węzeł **B** ma taki sam kolor jak poprzednio węzeł **A**. Zatem operacja rotacji przywróciła strukturę drzewa czerwono-czarnego.

**Algorytm wstawiania węzła do drzewa czerwono-czarnego**

**Wejście:**

|  |  |  |
| --- | --- | --- |
| *root* | – | referencja do zmiennej, która przechowuje wskazanie korzenia drzewa czerwono-czarnego |
| *k* | – | klucz nowego węzła (mogą również występować dodatkowe dane, które chcemy przechowywać w węźle ) |
| *S* | – | wskazanie węzła strażnika |

**Wyjście:**

Drzewo czerwono-czarne z wstawionym nowym węzłem.

**Zmienne pomocnicze:**

|  |  |  |
| --- | --- | --- |
| *X* | – | wskazanie nowego węzła |
| *Y* | – | wskazanie węzła |
| rot\_L ( *root*, *x*  ) | – | rotacja w lewo względem *x* |
| rot\_R ( *root*, *x*  ) | – | rotacja w prawo względem *x* |

[**Lista kroków:**](https://eduinf.waw.pl/inf/alg/001_search/0001.php#prezentacja)

|  |  |  |
| --- | --- | --- |
| K01: | Utwórz nowy węzeł | tworzymy nowy węzeł |
| K02: | *X*  ← adres nowego węzła |  |
| K03: | ( *X*→*left*  ) ← *S* | liśćmi staje się węzeł strażnik |
| K04: | ( *X*→*right*  ) ← *S* |  |
| K05: | ( *X*→*up*  ) ← *root* |  |
| K06: | ( *X*→*key*  ) ← *k* |  |
| K07: | **Jeśli** ( *X*→*up*  ) ≠ *S*, **to idź** do kroku K10 | sprawdzamy, czy drzewo jest puste |
| K08: | *root*  ← *X* | jeśli tak, to nowy węzeł staje się jego korzeniem |
| K09: | **Idź do** K17 | przechodzimy do sprawdzania warunków drzewa czerwono-czarnego |
| K10: | **Jeśli** *k*  < ( *X*→*up*→*key*  ), **to idź** do kroku K14 | porównujemy klucze |
| K11: | **Jeśli** ( *X*→*up*→*right*  ) = *S*, **to**: ( *X*→*up*→*right*  ) ← *X*      i**idź** do kroku K17 | jeśli prawy syn nie istnieje ( jest nim strażnik ), to nowy węzeł staje się prawym synem i przechodzimy do sprawdzania warunków drzewa czerwono-czarnego |
| K12: | ( *X*→*up*  ) ← ( *X*→*up*→*right*  ) | inaczej przechodzimy do prawego syna |
| K13: | **Idź** do kroku K10 | i kontynuujemy pętlę |
| K14 | **Jeśli** ( *X*→*up*→*left*  ) = *S*, **to**: ( *X*→*up*→*left*  ) ← *X*     i **idź** do kroku K17 | to samo dla lewego syna |
| K15: | ( *X*→*up*  ) ← ( *X*→*up*→*left*  ) |  |
| K16: | **Idź** do kroku K10 |  |
| K17: | ( *X*→*color*  ) ← RED | węzeł kolorujemy na czerwono |
| K18: | **Dopóki** ( *X*  ≠ *root*  ) obrazek( *X*→*up*→*color*  ) = RED ), **wykonuj** kroki K19..K47. | w pętli sprawdzamy kolejne przypadki |
| K19: | **Jeśli** ( *X*→*up*  ) ≠ ( *X*→*up*→*up*→left ),     **to idź do** kroku K34 | skok do przypadków lustrzanych |
| K20: | *Y*  ← ( *X*→*up*→*up*→*right*  ) | Y wskazuje wujka węzła X |
| K21: | **Jeśli** ( *Y*→*color*  ) = BLACK,     **to idź** do kroku K27 | sprawdzamy przypadek nr 1 -> wujek jest czerwony |
| K22: | ( *X*→*up*→*color*  ) ← BLACK | ojca kolorujemy na czarno |
| K23: | ( *Y*→*color*  ) ← BLACK | wujka kolorujemy na czarno |
| K24: | ( *X*→*up*→*up*→*color*  ) ← RED | dziadka kolorujemy na czerwono |
| K25: | *X*  ← ( *X*→*up*→*up*  ) | za nowe X przyjmujemy dziadka |
| K26: | **Następny obieg pętli** K18 | i kontynuujemy pętlę |
| K27: | **Jeśli** *X*  ≠ ( *X*→*up*→right ),     **to idź** do kroku K30 | sprawdzamy przypadek 2 -> wujek czarny, X prawy syn |
| K28: | *X*  ← ( *X*→*up*  ) | X staje się swoim ojcem |
| K29: | rot\_L ( *root*, *X*  ) | wykonujemy rotację w lewo, X przechodzi do lewego syna |
| K30: | ( *X*→*up*→*color*  ) ← BLACK | przypadek 3 -> wujek czarny, X lewy syn |
| K31: | ( *X*→*up*→*up*→*color*  ) ← RED | zmieniamy kolory ojca i dziadka |
| K32 | rot\_R ( *root*, ( *X*→*up*→*up*  ) ) | wykonujemy rotację w prawo względem dziadka |
| K33: | **Idź** do kroku K48 | i wychodzimy z pętli |
| K34: | *Y*  ← ( *X*→*up*→*up*→*left*  ) | przypadki lustrzane |
| K35: | **Jeśli** ( *Y*→*color*  ) = BLACK,     **to idź** do kroku K41 |  |
| K36: | ( *X*→*up*→*color*  ) ← BLACK |  |
| K37: | ( *Y*→*color*  ) ← BLACK |  |
| K38: | ( *X*→*up*→*up*→*color*  ) ← RED |  |
| K39: | *X*  ← ( *X*→*up*→*up*  ) |  |
| K40: | **Następny obieg pętli** K18 |  |
| K41: | **Jeśli** *X*  ≠ ( *X*→*up*→left ),     **to idź** do kroku K44 |  |
| K42: | *X*  ← ( *X*→*up*  ) |  |
| K43: | rot\_R ( *root*, *X*  ) |  |
| K44: | ( *X*→*up*→*color*  ) ← BLACK |  |
| K45: | ( *X*→*up*→*up*→*color*  ) ← RED |  |
| K46 | rot\_L ( *root*, ( *X*→*up*→*up*  ) ) |  |
| K47: | **Idź** do kroku K48 | wychodzimy z pętli |
| K48 | ( *root*→*color*  ) ←  BLACK | kolorujemy korzeń na czarno |
| K49: | **Zakończ** |  |