Parcours d'un graphe

A la base des algorithmes sur les graphes, on trouve les parcours de graphe, c'est à dire l'exploration des sommets. A partir du sommet de départ, on peut :

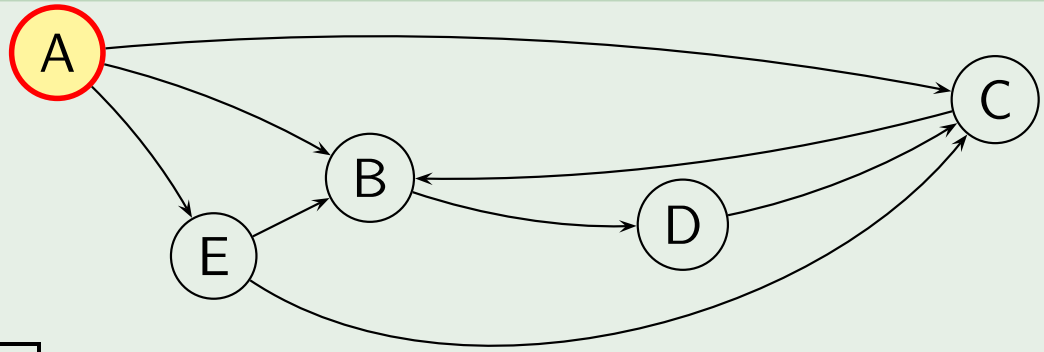
- explorer tous ses voisins immédiats, puis les voisins des voisins et ainsi de suite. Le graphe est donc exploré en « cercle concentrique »autour du sommet de départ . . ., on parle alors de parcours en largeur ou breadth first search (*BFS*) en anglais.

Parcours d'un graphe

A la base des algorithmes sur les graphes, on trouve les parcours de graphe, c'est à dire l'exploration des sommets. A partir du sommet de départ, on peut :

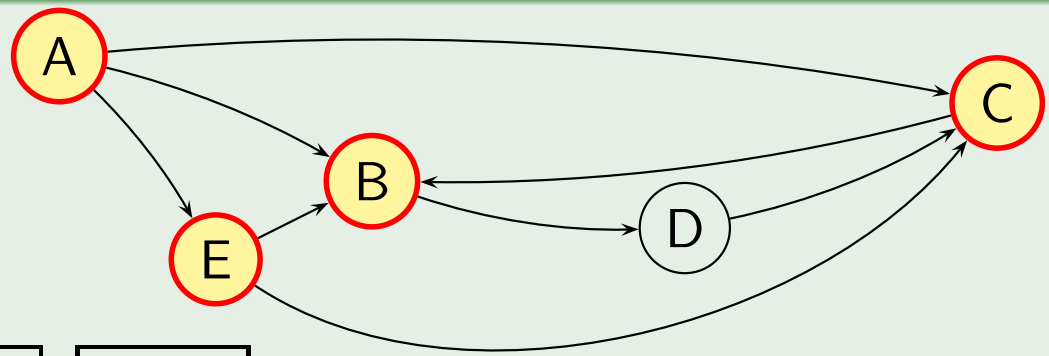
- explorer tous ses voisins immédiats, puis les voisins des voisins et ainsi de suite. Le graphe est donc exploré en « cercle concentrique »autour du sommet de départ . . ., on parle alors de parcours en largeur ou breadth first search (*BFS*) en anglais.
- explorer à chaque étape le premier voisin non encore exploré. Lorsque qu'on atteint un sommet dont tous les voisins ont déjà été exploré, on revient en arrière, on parle alors de parcours en profondeur ou depth first search (*DFS*) en anglais.

Exemple de parcours en largeur



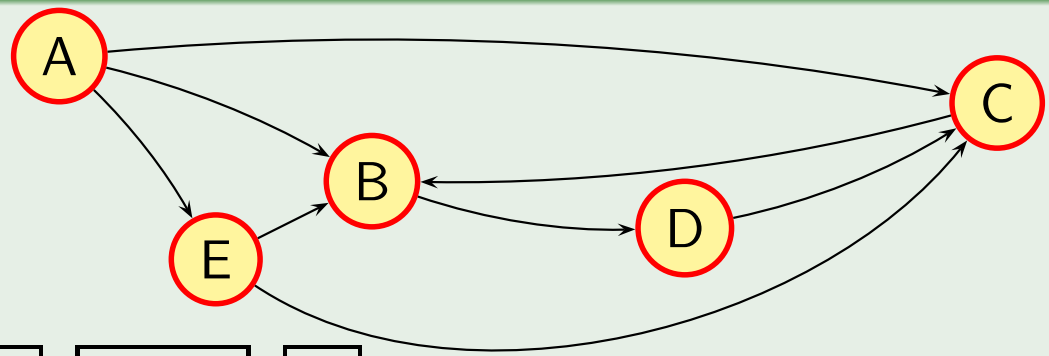
Sommets explorés : A

Exemple de parcours en largeur



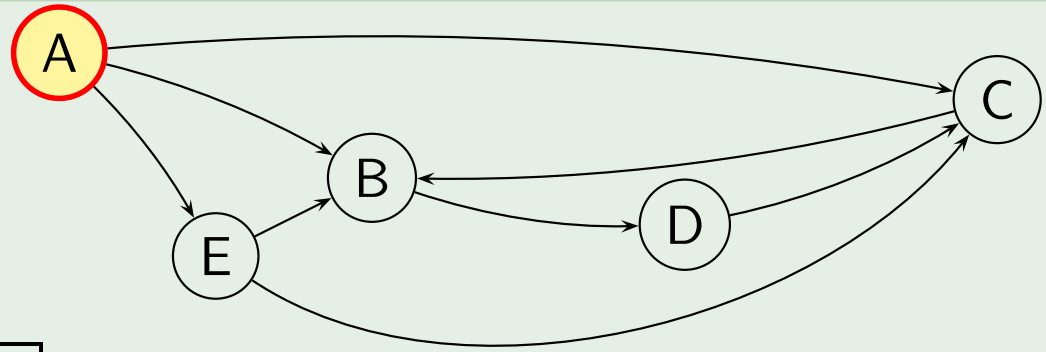
Sommets explorés : A, B,C,E

Exemple de parcours en largeur



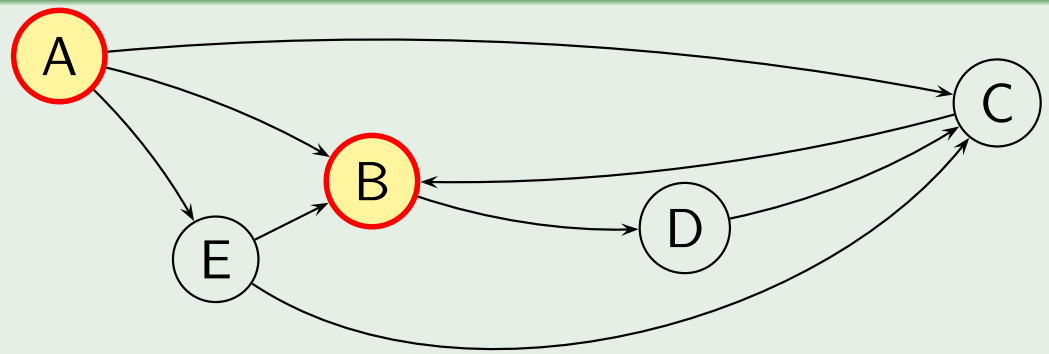
Sommets explorés : A, B,C,E, D.

Exemple de parcours en profondeur



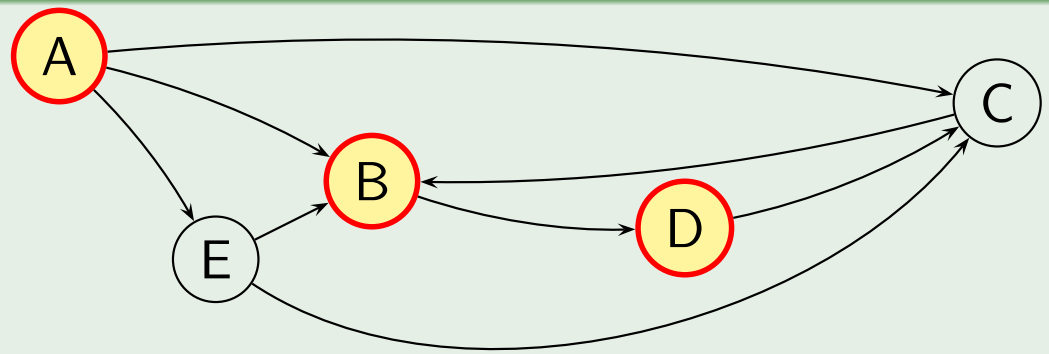
Sommets explorés : A

Exemple de parcours en profondeur



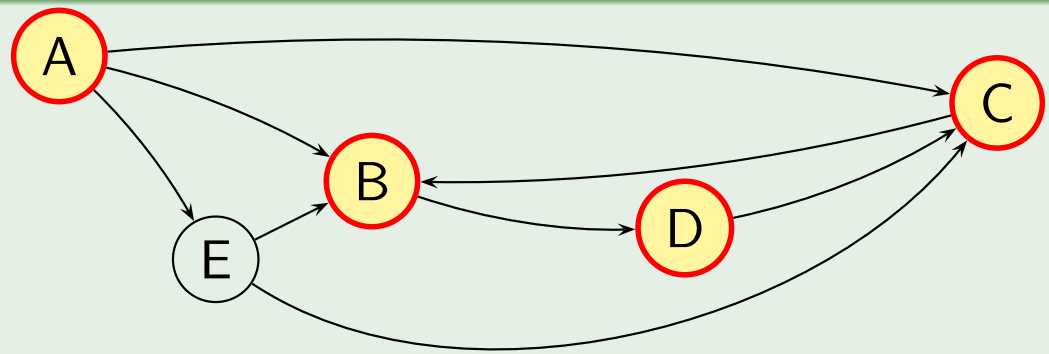
Sommets explorés : A, B

Exemple de parcours en profondeur



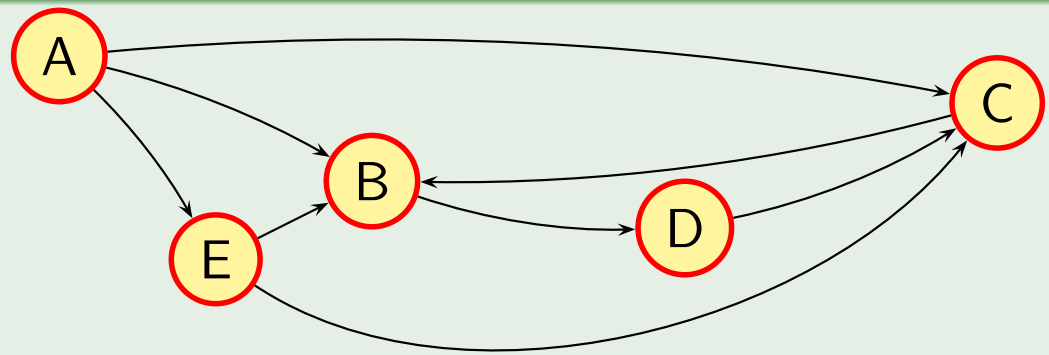
Sommets explorés : A, B, D

Exemple de parcours en profondeur



Sommets explorés : A, B, D, C

Exemple de parcours en profondeur



Sommets explorés : A, B, D, C, E

File et parcours en largeur

- Pour un parcours en largeur, on doit stocker dans une structure de données les sommets en attente d'être explorés. C'est à dire les voisins du sommet de départ, puis les voisins des voisins . . .Ces sommets doivent être retirés pour exploration, dans leur ordre d'insertion, la structure de données utilisée est donc du type premier entré, premier sorti (first in first out (*FIFO*)), c'est donc une file.

File et parcours en largeur

- Pour un parcours en largeur, on doit stocker dans une structure de données les sommets en attente d'être explorés. C'est à dire les voisins du sommet de départ, puis les voisins des voisins . . .Ces sommets doivent être retirés pour exploration, dans leur ordre d'insertion, la structure de données utilisée est donc du type premier entré, premier sorti (first in first out (*FIFO*)), c'est donc une file.

File et parcours en largeur

- Pour un parcours en largeur, on doit stocker dans une structure de données les sommets en attente d'être explorés. C'est à dire les voisins du sommet de départ, puis les voisins des voisins . . .Ces sommets doivent être retirés pour exploration, dans leur ordre d'insertion, la structure de données utilisée est donc du type premier entré, premier sorti (first in first out (*FIFO*)), c'est donc une file.
- Pour l'implémentation on doit pouvoir enfiler (ajouter un sommet dans la file) et défiler (retirer une sommet) de façon efficace donc en $O(1)$.

File et parcours en largeur

- Pour un parcours en largeur, on doit stocker dans une structure de données les sommets en attente d'être explorés. C'est à dire les voisins du sommet de départ, puis les voisins des voisins . . .Ces sommets doivent être retirés pour exploration, dans leur ordre d'insertion, la structure de données utilisée est donc du type premier entré, premier sorti (first in first out (*FIFO*)), c'est donc une file.
- Pour l'implémentation on doit pouvoir enfiler (ajouter un sommet dans la file) et défiler (retirer une sommet) de façon efficace donc en $O(1)$.
- Pour l'implémentation on pourra utiliser

File et parcours en largeur

- Pour un parcours en largeur, on doit stocker dans une structure de données les sommets en attente d'être explorés. C'est à dire les voisins du sommet de départ, puis les voisins des voisins . . .Ces sommets doivent être retirés pour exploration, dans leur ordre d'insertion, la structure de données utilisée est donc du type premier entré, premier sorti (first in first out (*FIFO*)), c'est donc une file.
- Pour l'implémentation on doit pouvoir enfiler (ajouter un sommet dans la file) et défiler (retirer une sommet) de façon efficace donc en $O(1)$.
- Pour l'implémentation on pourra utiliser
  - Le module `Queue` de OCaml

File et parcours en largeur

- Pour un parcours en largeur, on doit stocker dans une structure de données les sommets en attente d'être explorés. C'est à dire les voisins du sommet de départ, puis les voisins des voisins . . .Ces sommets doivent être retirés pour exploration, dans leur ordre d'insertion, la structure de données utilisée est donc du type premier entré, premier sorti (first in first out (*FIFO*)), c'est donc une file.
- Pour l'implémentation on doit pouvoir enfiler (ajouter un sommet dans la file) et défiler (retirer une sommet) de façon efficace donc en $O(1)$.
- Pour l'implémentation on pourra utiliser
  - Le module `Queue` de OCaml
  - Une structure de liste chaînée avec des opérations enfiler et défiler en $O(1)$

File et parcours en profondeur

- Pour un parcours en profondeur, on stocke aussi dans une structure de données les sommets en attente d'être explorés. Mais cette fois, la structure de données utilisée est du type dernier entré, premier sorti (last in first out (*LIFO*)), c'est à dire une pile.

File et parcours en profondeur

- Pour un parcours en profondeur, on stocke aussi dans une structure de données les sommets en attente d'être explorés. Mais cette fois, la structure de données utilisée est du type dernier entré, premier sorti (last in first out (*LIFO*)), c'est à dire une pile.

File et parcours en profondeur

- Pour un parcours en profondeur, on stocke aussi dans une structure de données les sommets en attente d'être explorés. Mais cette fois, la structure de données utilisée est du type dernier entré, premier sorti (last in first out (*LIFO*)), c'est à dire une pile.
- Pour l'implémentation, on peut :

File et parcours en profondeur

- Pour un parcours en profondeur, on stocke aussi dans une structure de données les sommets en attente d'être explorés. Mais cette fois, la structure de données utilisée est du type dernier entré, premier sorti (last in first out (*LIFO*)), c'est à dire une pile.
- Pour l'implémentation, on peut :
  - se contenter d'utiliser la récursivité de façon à ce que la pile des sommets en attente d'être exploré soit gérée de façon automatique via la pile des appels récursifs.

File et parcours en profondeur

- Pour un parcours en profondeur, on stocke aussi dans une structure de données les sommets en attente d'être explorés. Mais cette fois, la structure de données utilisée est du type dernier entré, premier sorti (last in first out (*LIFO*)), c'est à dire une pile.
- Pour l'implémentation, on peut :
  - se contenter d'utiliser la récursivité de façon à ce que la pile des sommets en attente d'être exploré soit gérée de façon automatique via la pile des appels récursifs.
  - Utiliser le module `Stack` de OCaml (ou une simple liste).

File et parcours en profondeur

- Pour un parcours en profondeur, on stocke aussi dans une structure de données les sommets en attente d'être explorés. Mais cette fois, la structure de données utilisée est du type dernier entré, premier sorti (last in first out (*LIFO*)), c'est à dire une pile.
- Pour l'implémentation, on peut :
  - se contenter d'utiliser la récursivité de façon à ce que la pile des sommets en attente d'être exploré soit gérée de façon automatique via la pile des appels récursifs.
  - Utiliser le module `Stack` de OCaml (ou une simple liste).
  - Utiliser une liste chainée en C afin d'implémenter une pile.