**Projet Info LDD2**

Modèle d’état des lieux du projet à compléter

Noms, prénoms des membres du groupe : Yehor KOROTENKO, Sipan BAREYAN, Ivan KHARKOV

→ Expliciter l’approche choisie et l’état actuel pour chaque thème :

Fonctionnel / fonctionnel sur certains exemples / non fonctionnel

→ Les erreurs et points à améliorer sont à détailler dans la colonne Remarques.

→ Dans Remarque, préciser si les question bonus ont été abordées.

Tous les TD ont été réalisés dans leur intégralité et testés exhaustivement. Toutes les questions bonus ont été abordées, à l’exception de celle du dernier TD.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TD** | **Thèmes** | **Code : classe / méthode** | **Tests** | **Remarques** |
| 1 | Implémentation de graphes, manipulation | node (constructeur, getters/setters, gestion multiplicité), open\_digraph.\_\_init\_\_, getters (get\_nodes, get\_nodes\_ids, …), new\_id, add\_edge, add\_edges | **node\_test.py**, **open\_digraph\_test.py** (init, copy, new\_id, getters, setters, add\_edge) | État : Fonctionnel |
| 2 | remove (parent, edge…), is\_wxell\_formed, add\_input\_node, add\_output\_node | Graph : remove\_parent, remove\_edge, is\_well\_formed, add\_input\_node, add\_output\_node | **open\_digraph\_test.py** (remove\_edge, remove\_node\_by\_id, add\_input\_node, add\_output\_node, well-formedness) | État : Fonctionnel |
| 3 | Traductions graphe/matrice d'adjacence ; génération  aléatoire de matrice / graphe | graph\_to\_adj\_matrix, adj\_matrix\_to\_graph, random\_adjacency\_matrix, random\_graph | **matrix\_test.py** (contraintes des matrices, from\_matrix, random) | État : Fonctionnel  Bonus : abordées |
| 4 | Sauvegarde (graphe vers .dot ; .dot vers graph) et représentation de graphes. Affichage (display) | to\_dot, from\_dot, display\_graph | **matrix\_test.py** (lecture/écriture .dot, adjacancy\_matrix invariance) | État : Fonctionnel |
| 5 | Classe pour les circuitds booléens, méthode is\_cyclic, is\_well\_formed de  bool\_circ, shift\_indices | BoolCircuit : is\_cyclic, is\_well\_formed, shift\_indices | **bool\_circ\_test.py** (acylcicité, validité de circuits, mode debug, circuits vides pour non-well-formed) | État : Fonctionnel |
| 6 | Compositions & connectivité ; iparallel, icompose, identity, connected\_components d'open\_digraph | open\_digraph : iparallel, icompose, identity, connected\_components | **open\_digraph\_test.py** (iparallel, parallel, icompose, compose), **TestGraphOperations** (connected\_components, split) | État : Fonctionnel |
| 7 | Longueur de chemins : l'algorithme de Dijkstra, shortest\_path, associer à chaque ancêtre commun des deux nœuds sa  distance à chacun des deux nœuds | open\_digraph\_mixins/open\_digraph\_dijkstra\_shortest\_mixin.py : dijkstra, shortest\_path, common\_ancestors | tests/path\_test.py : test\_dijkstra (bidirectionnel, parents/children), test\_common\_ancestors | État : Fonctionnel |
| 8 | Tri topologique, chemin le plus long, mixins. | open\_digraph\_mixins/open\_digraph\_longest\_mixin.py : sort\_topologicly, get\_node\_depth, get\_graph\_depth, longest\_path | tests/path\_test.py : test\_topological\_sort, test\_node\_depth, test\_graph\_depth, test\_longest\_paht, test\_cyclic\_raises\_exception | État : Fonctionnel |
| 9 | Synthèse de circuit via une formule propositionnelle. | modules/bool\_circ.py : fonction parse\_parentheses(...) (parser naïf de formules), build\_adder\_0 (construction directe depuis la formule) | tests/additioner\_test.py : test\_build\_adder\_0\_basic\_structure (inputs/outputs, bien-formé) - tests/parentheses\_test.py (vérifie fuse\_nodes utilisé après parsing) | État : Fonctionnel  Bonus : abordées |
| 10 | Circuits booléens aléatoires ; Additionneur. | modules/bool\_circ.py : random\_bool\_circ\_from\_graph, from\_number, add\_two\_numbers, build\_adder, build\_adder\_inner, build\_half\_adder | - tests/additioner\_test.py : test\_evaluate\_of\_additioner, test\_add\_some\_nums test\_build\_half\_adder\_alias, test\_build\_adder\_dimensions, test\_build\_adder\_invalid\_arguments | État : Fonctionnel  Bonus : abordées |
| 11 | Réécriture : évaluation de circuits. | modules/bool\_circ.py : méthode evaluate + toutes les transform\_\* (copy, not, and/or/xor, associativité, involution, erase, etc.) | - tests/bool\_circ\_test.py : ~25 tests couvrant chaque règle : test\_constant\_copy\_transform, test\_constant\_not\_transform, test\_transform\_and\_zero, …, test\_transform\_not\_involution, test\_evaluate\_simple, test\_transform\_associative\_xor(\_2), test\_transform\_associative\_copy, test\_transform\_involution\_xor(\_odd,\_dont\_work), test\_transform\_erase\_operator, test\_transform\_xor\_if\_has\_parent\_not, test\_transform\_copy\_if\_has\_parent\_not… | État : Fonctionnel |
| 12 | Réécriture : vérification d'un code de Hamming. | bool\_circ.py :  generate\_4bit\_encoder, generate\_4bit\_decoder, carry\_lookahead\_4(n), carry\_lookahead\_4n, get\_result\_of\_evaluated\_enc\_dec | tests/encoder\_decoder\_test.py : test\_encoder\_deconder\_compose\_evaluate\_gives\_identity (et variantes), vérifie qu’après encodage+décodage et evaluate on récupère les bits d’origineb | État : Fonctionnel |