Préprojet Prolog 2011/2012

Julien Reichert

LSV, ENS Cachan

Vendredi 17 février 2012

Voici les quatre exercices qui vous seront donnés en guise de préprojet. Le barème est le suivant : meilleur exercice sur 5, deuxième sur 8, troisième sur 6 et quatrième sur 3. Les critères de notation seront lisibilité, concision, commentaires... et naturellement il faut que cela fonctionne !

Consignes

Le préprojet sera rendu sous forme d'archive contenant 4 fichiers judicieusement nommés correspondant à chacun des exercices. Tout fichier supplémentaire en-dehors d'exemples nommés tout aussi judicieusement devra être présenté brièvement dans le message livrant le préprojet (qui doit être envoyé à l'adresse `reichert@lsv.ens-cachan.fr` avant le vendredi 9 mars 2012 à 0 h 42) en justifiant si possible son utilité. Il peut s'agir par exemple d'un programme générant des entrées pour l'un des exercices, auquel cas le langage est plus ou moins libre, mais la notation ne tiendra pas compte dudit programme.

1 Algorithme de Dijkstra

Implémentez l'algorithme de Dijkstra en Prolog ; posez-vous bien auparavant la question de la représentation d'un arc pondéré.

Indication : 10 lignes de code (sans compter les sauts de ligne, les commentaires et la gestion de l'entrée) suffisent.

2 Sudoku et variantes

Créez un prédicat résolvant le sudoku, en utilisant si possible les techniques de programmation par contraintes que vous n'avez pas encore apprises. Les nombres donnés à la base devront pouvoir être saisis à la main selon la syntaxe de votre choix. Ensuite traitez les variantes suivantes :

– kenken, où il n'y a plus les neuf régions carrées de neuf cases, mais des régions de taille variable où figure une contrainte arithmétique, par exemple la somme ou le produit des nombres de la région vaut X, la différence ou le rapport des **deux** nombres de la région vaut X dans un certain ordre.
– unequal, où là encore les régions n'apparaissent pas, mais à la place certains bords entre deux cases adjacentes ont une indication indiquant quelle case a le plus grand numéro.
– adjacent, dont le principe est proche d'unequal à ceci près que toutes les cases ont un marqueur, qui indique si la différence entre les numéros est égale à ± 1 ou non.
– towers, où cette fois on vous donne comme indication pour certaines lignes la taille de la sous-suite croissante minimale dans l'ordre lexicographique des indices. Il s'agit en fait de voir les cases comme des tours de hauteur leur contenu. L'indication est alors le nombre de tours visibles de face. Par exemple si les 6 cases sont de gauche à droite $2, 5, 4, 3, 6, 1$, les indications sont 3 à gauche et 2 à droite.

Si vous trouvez que votre code est très long, pas de panique, les contraintes sont effectivement nombreuses et prennent du temps. Je rappelle que la programmation par copier-coller, c'est **mal** ! (Mais ici ça fonctionne bien.)

3 Ouverture au bridge

Sur la page du projet figure un lien vers le fichier expliquant le système d'enchères français au bridge. Créez un prédicat ouverture prenant en entrée la main de départ et donnant l'ouverture recommandée. L'important n'est pas de gérer au mieux les cas difficiles mais de jongler avec les prédicats annexes et d'établir les priorités. Pensez à trouver un moyen de générer des donnes plus agréablement, vous me mettrez dans de meilleures dispositions.

Indication : considérer la majorité des cas risque de vous prendre environ 200 lignes. Définissez la notion de levée de jeu comme vous le voulez, une bonne approximation est la somme pour chaque couleur de (nombre d'honneurs à la couleur $-$ 4) + (nombre de cartes à la couleur). Une main régulière est une main n'ayant aucune couleur d'une ou aucune carte (on tolère un seul carreau ou trèfle en pratique) et au plus deux couleurs de deux cartes, en outre il faut qu'il y ait au plus quatre cartes à pique ainsi qu'à cœur. Vous aurez besoin de prédicats auxiliaires tels que : `points_honneur(`$Main$`, `N`)`, `nombre(`$Main$`, `$Couleur$`, `N`)` éventuellement en quatre prédicats, `distribution_reguliere(`$Main$`)`, `unicolore(`$Main$`)` idem concernant la couleur (signifie au moins 6 cartes), `levees_jeu(`$Main$`, `N`)` selon la définition annoncée plus haut, `trois_honneurs_sur_cinq(`$Main$`, `$Couleur$`)` et d'autres, suivant les besoins, servant à composer les différents cas du prédicat final `ouverture(`$Main$`, `$Ouverture$`)` (n'hésitez pas à utiliser `write`) où $Ouverture$ peut être n'importe quelle enchère (hors contres) du bridge, à savoir `passe` ou {`un`, `deux`, `trois`, `quatre`, `cinq`, `six`, `sept`}{`T`, `K`, `C`, `P`, `SA`}, même si normalement les ouvertures au-delà de `cinqK` n'ont pas lieu d'être.

4 Chemin hamiltonien

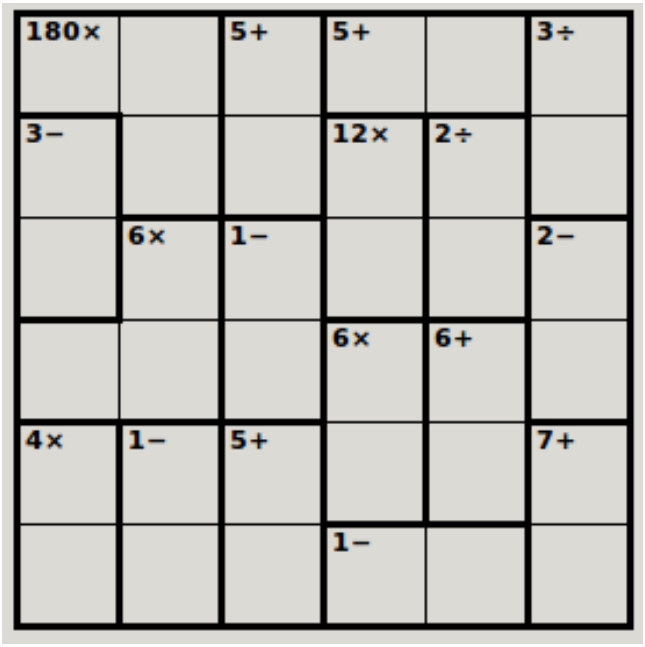
À partir du plan des 16 lignes du métro parisien, décidez s'il y existe un chemin hamiltonien. On définit ici un chemin hamiltonien comme un trajet passant par toutes les lignes une et une seule fois. Il n'est donc pas nécessaire de passer par toutes les stations. En outre, tentez d'interdire les correspondances multiples à une seule station si vous voulez la totalité des points.

Indication : 12 lignes de code suffisent, auxquelles on ajoute le contenu de `metro.prolog`.

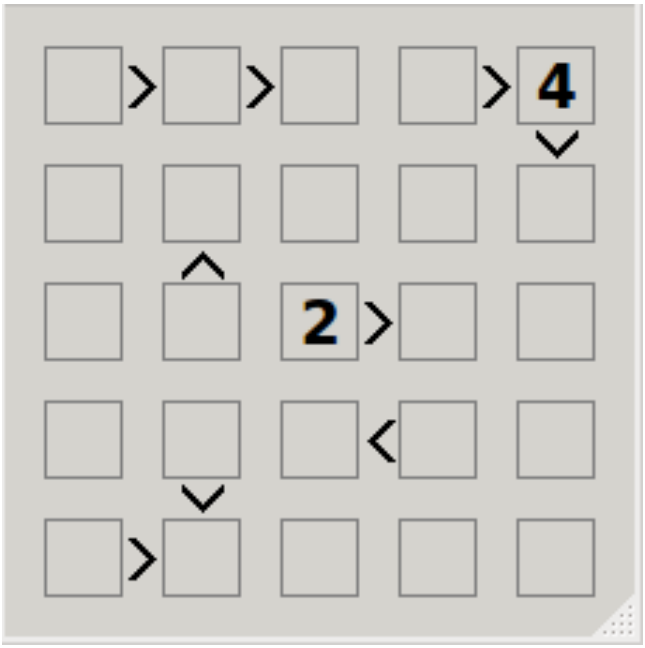
Annexe

Sudoku

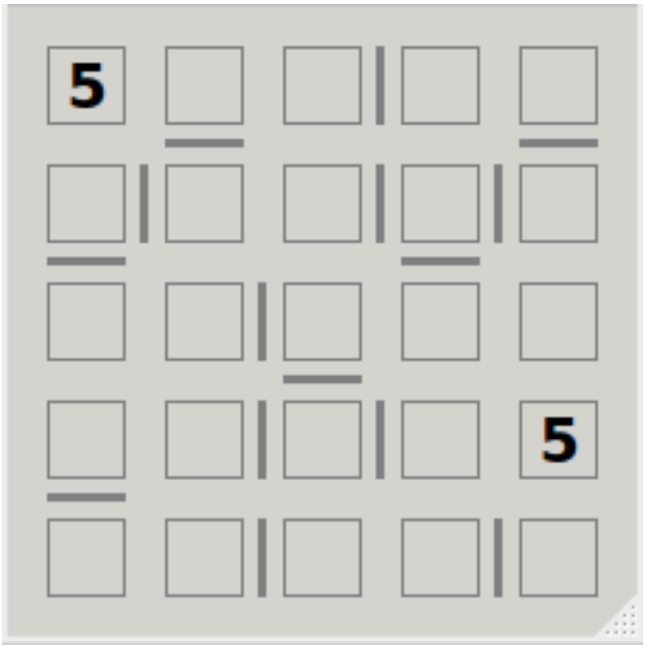
Pour vous donner une idée du kenken, une grille de départ figure ci-dessous :



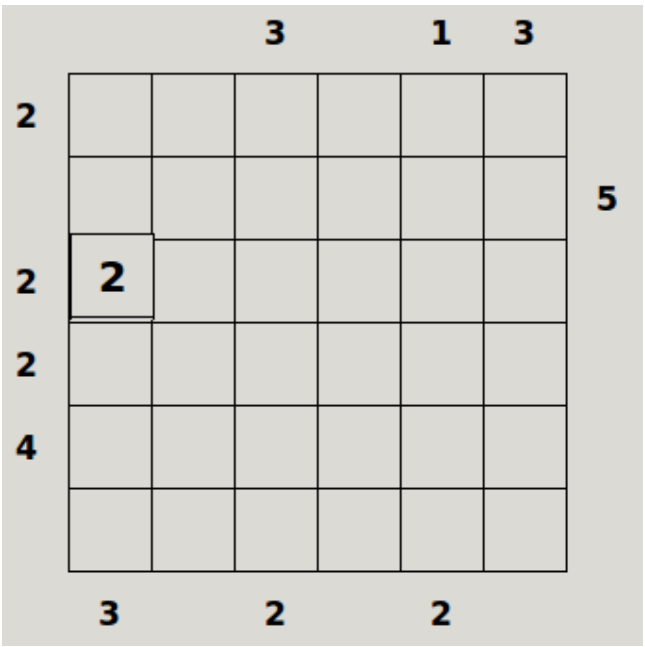
idem pour unequal :



adjacent :



et towers :



Bridge

Votre programme devrait retourner pour les mains suivantes (en cas de différence, posez-moi la question, il y a parfois une alternative défendable) :

- Pique : R D V 3 - Cœur : D V 8 - Carreau : 4 - Trèfle : A R 10 6 2 → **1 TREFLE**
- Pique : R D V 3 2 - Cœur : V 8 - Carreau : 4 - Trèfle : A R 10 6 2 → **1 PIQUE**
- Pique : R D V - Cœur : D V 8 - Carreau : 10 4 - Trèfle : A R 10 6 2 → **1 SANS ATOUT**
- Pique : R D V 3 - Cœur : D V 8 4 3 2 - Carreau : rien - Trèfle : 10 6 2 → **PASSE** (**2 CŒUR** discutable)
- Pique : R D V 10 6 3 2 - Cœur : D V 8 - Carreau : 4 - Trèfle : 10 9 → **3 PIQUE**
- Toutes les cartes à Pique : **2 CARREAU**, et pourquoi pas **7 PIQUE** (moins rentable en pratique, il vaut mieux y arriver progressivement)

Chemin hamiltonien

La réponse est oui, voici un exemple :

- Ligne 1 N'importe où ⟶ Ligne 5 Bastille
- Ligne 5 Bastille ⟶ Ligne 10 Gare d'Austerlitz
- Ligne 10 Gare d'Austerlitz ⟶ Ligne 13 Duroc
- Ligne 13 Duroc ⟶ Ligne 12 Montparnasse Bienvenüe
- Ligne 12 Montparnasse Bienvenüe ⟶ Ligne 14 Saint-Lazare
- Ligne 14 Saint-Lazare ⟶ Ligne 11 Châtelet
- Ligne 11 Châtelet ⟶ Ligne 3bis Porte des Lilas
- Ligne 3bis Porte des Lilas ⟶ Ligne 3 Gambetta
- Ligne 3 Gambetta ⟶ Ligne 8 Opéra
- Ligne 8 Opéra ⟶ Ligne 4 Strasbourg Saint-Denis
- Ligne 4 Strasbourg Saint-Denis ⟶ Ligne 7 Gare de l'Est
- Ligne 7 Gare de l'Est ⟶ Ligne 6 Place d'Italie
- Ligne 6 Place d'Italie ⟶ Ligne 9 Trocadéro
- Ligne 9 Trocadéro ⟶ Ligne 2 Nation
- Ligne 2 Nation ⟶ Ligne 7bis Jaurès

Utilisez la fonction `write` pour imprimer le chemin hamiltonien trouvé et le vérifier sur le plan. Pour vous faciliter la vie, vous pouvez utiliser les lignes de code figurant sur la page du préprojet.