Programmation impérative (novice)

Projet

Licence informatique 2ème année

Université de La Rochelle

La Rochelle Université

2023-2024_2

Les lignes commençant par $ représente une commande *unix*. Le $ ne fait pas partie de la commande proprement dite mais symbolise l'invite de commande.

Table des matières

1 Consignes

1.1 Groupes

Le projet se fait par groupe de maximum 6 étudiants et est à rendre par un dépôt *moodle* le vendredi 8 décembre 2023 à 23h00. Un retard raisonnable est toléré mais il sera pénalisé.

L'utilisation des générateurs de code de type *copilot* est fortement encouragée.

1.2 Langue utilisée

La langue utilisée dans le code et la documentation devra être exclusivement l'anglais. Si vous avez des difficultés dans la langue de Shakespeare, vous pourrez utiliser les traducteurs automatiques :

- https://www.deepl.com/translator
- https://translate.google.fr/
- https://chat.openai.com/

1.3 Nommage

1.3.1 Fichiers

- les noms de fichiers du langage C seront tous en minuscules et en anglais. S'ils sont composés de plusieurs mots, ils devront être séparés par un tiret (−) (notation https://en.wikipedia.org/wiki/Letter_case#Kebab_case);
- les fichiers contenant le code devront avoir l'extension `.c` ;
- les fichiers d'en-têtes (exportables) devront avoir l'extension `.h` ;
- les fichiers destinés à être inclus dans votre code mais non exportables devront avoir l'extension `.inc`

1.3.2 Types

Les noms de types devront faire commencer chaque mot qui les compose par une majuscule. Il n'y a pas de sous-tirets (notation https://en.wikipedia.org/wiki/Letter_case#Camel_case). Les structures devront commencer par un sous-tiret (_) puis suivre la notation *Camel case* pour ne pas les confondre avec les noms de types.

1.3.3 Macros

Les macros (avec ou sans arguments) s'écrivent tout en majuscule en séparant les mots par des sous-tirets (_) (notation https://en.wikipedia.org/wiki/Letter_case#Snake_case).

1.3.4 Variables et fonctions

Les variables et les fonctions s'écrivent toutes en minuscules en séparant les mots par des sous-tirets (notation https://en.wikipedia.org/wiki/Letter_case#Snake_case).

1.4 Style

1.4.1 Marque d'inclusion unique

Chaque fichier d'en-tête devra posséder une marque permettant d'éviter les conséquences d'un fichier inclus plusieurs fois. Voir https://google.github.io/styleguide/cppguide.html#The__define_Guard

1.4.2 Ordre des inclusions

L'inclusion des fichiers d'en-tête devra respecter la logique suivante :

1. Inclusion du fichier directement lié au fichier `.c` qui l'inclut suivi d'une ligne vide ;
2. inclusion des fichiers d'en-tête du C standard suivis d'une ligne vide ;
3. inclusion des fichiers d'en-tête provenant d'autres librairies suivis d'une ligne vide ;
4. inclusion des fichiers d'en-tête du projet suivi d'une ligne vide ;
5. inclusion des fichiers d'inclusion (extension `.inc`)

1.4.3 Indentation

Le style d'indentation devra être celui préconisé par Google https://google.github.io/styleguide/cppguide.html#Formatting. L'utilitaire `clang-format` (https://clang.llvm.org/docs/ClangFormat.html) supporte le style `Google`.

Vous pourrez utiliser l'utilitaire `cclint` pour vérifier votre code.

1.5 Structuration du code

Les champs des structures seront protégés à la manière de la librairie `fraction` vue en travaux pratiques. Un soin sera tout particulièrement apporté à la structuration du code notamment en ce qui concerne les structures de données utilisées.

1.6 Documentation

La documentation sera générée avec l'outil `sphinx` et les fonctions seront documentées avec la norme de `doxygen`.

1.7 Tests

Des tests unitaires devront être implémentés, ils testeront chaque fonction et s'efforceront de vérifier que la mémoire est bien libérée au moyen de l'utilitaire `valgrind`.

Vous pourrez vous inspirer du projet https://github.com/chdemko/c-arithmetic.

D'une manière générale, toutes les options possibles décrites dans ce projet devront être implémentées.

2 Sujet

Le but du projet est de produire un exécutable permettant de générer des emplois du temps à l'aide des algorithmes génétiques. Plusieurs librairies (compilées sur les machines virtuelles de l'Université) sont fournies permettant de vous concentrer sur la problématique d'emplois du temps :

- `bitset` permettant de gérer indirectement des ensembles d'entiers ;
- `genetic-algorithm` permettant de gérer la recherche de solution via la technologie des algorithmes génétiques ;
- `spacetimeslot` permettant de gérer les créneaux disponibles pour une salle ou un groupe de salles ;
- `teachingslot` permettant de gérer les intervenants d'un cours (étudiants, enseignants, techniciens, ...).

Ces librairies doivent obligatoirement être utilisées dans votre projet.

Le projet pourra fournir un fichier `README.md` expliquant les instructions à exécuter avant de le configurer.

Le projet devra fournir une documentation produite avec

```
$ make docs
```

Il pourra être installé avec

```
$ make install
```

2.1 Description du problème (produit par chatGPT)

La génération d'un emploi du temps est une tâche complexe qui consiste à créer un plan horaire pour un individu ou un groupe d'individus, en attribuant des activités, des tâches ou des cours à des plages horaires spécifiques sur une période donnée. Cette problématique se pose dans de nombreux contextes, tels que l'éducation (emplois du temps scolaires), le monde professionnel (emplois du temps pour les employés), les événements, les conférences, les transports en commun, etc.

La principale problématique de génération d'un emploi du temps réside dans l'optimisation des ressources et la satisfaction des contraintes imposées, tout en tenant compte de divers facteurs tels que les préférences des individus, les ressources disponibles, les contraintes de temps, les priorités, les compétences nécessaires, etc. Cela peut devenir un véritable casse-tête lorsque de nombreuses variables interagissent.

Voici quelques-unes des contraintes et des objectifs typiques associés à la génération d'un emploi du temps :

1. Contraintes de disponibilité : Il faut tenir compte des heures de travail, des horaires de cours, des plages horaires de disponibilité des individus ou des ressources.
2. Équilibrage des charges de travail : Répartir équitablement les tâches ou les cours sur une période donnée pour éviter la surcharge ou la sous-utilisation de ressources.

3. Préférences individuelles : Prendre en considération les préférences personnelles des individus, comme les préférences d'horaires ou de lieu.
4. Minimisation de conflits : Éviter les conflits d'horaires entre les activités ou les cours pour une même personne ou ressource.
5. Utilisation efficace des ressources : Optimiser l'utilisation des ressources disponibles tout en minimisant les coûts.
6. Respect des contraintes légales ou règlementaires : Se conformer aux lois ou aux règlements qui imposent des limites aux heures de travail ou aux heures de cours.
7. Maximisation de la productivité ou de l'efficacité : Organiser les activités de manière à maximiser la productivité, le rendement ou l'efficacité.
8. Réactivité aux changements : La capacité à ajuster rapidement l'emploi du temps en cas de perturbations imprévues.

La génération d'un emploi du temps peut être effectuée manuellement, mais elle est de plus en plus automatisée grâce à des outils informatiques et des algorithmes d'optimisation. Ces algorithmes tentent de résoudre ces problèmes complexes en prenant en compte toutes les contraintes et objectifs pour générer un emploi du temps optimal ou satisfaisant, en fonction des critères définis.

2.2 Modélisation

Un problème d'emploi du temps peut être décrit par la description de toutes les activités à programmer. Chaque activité est décrite par :

- un titre (e.g. « CM de Programmation impérative (novice) »)
- les groupes de participants concernés (groupe d'étudiants, groupe d'enseignants, groupe de techniciens, …)
- un ensemble de créneaux spatio-temporels possibles (e.g. « Salle PAS000 le lundi de 9h45 à 11h15 », « Salle PAS000 le lundi de 11h30 à 13h00 », …). Pour des raisons techniques, la salle d'un créneau spatio-temporel sera représentée par un groupe de salles (réduit la plupart des cas à un singleton).

Résoudre l'emploi du temps consiste à affecter à chaque activité l'un des créneaux spatio-temporels possibles tout en minimisant les contraintes violées :

- contraintes de conflits : une personne ne peut être présente dans deux activités au même moment, une salle ne peut pas être utilisée par deux activités au même moment (ces contraintes sont appelées **contraintes fortes**) ;
- contraintes de précédence : une activité doit être programmée avant une autre (**contraintes faibles**) ;
- contraintes de proximité : une activité doit être programmée le plus près d'une autre (**contraintes faibles**) ;
- contraintes d'occupation : essayer de regrouper les créneaux occupés de chaque salle (**contraintes faibles**) ;
- contraintes d'efficacité : essayer de regrouper les créneaux de chaque individu (éviter les « trous » au sein d'une même journée) (**contraintes faibles**) ;

- contraintes de répartition : essayer de répartir les activités d'un individu sur l'ensemble des jours disponibles (**contraintes faibles**).

Le problème sera décrit par un fichier au format *YAML* (voir le fichier `problem.yaml` joint pour un exemple) comportant :

- la liste des salles possibles
- la liste des personnes concernées
- la liste des groupes construits (tant au niveau des salles que des personnes)
- la liste des activités
- la liste des contraintes autres que les contraintes fortes

D'un point de vue technique, on pourra considérer les salles et les personnes comme des ressources (inutile de les différencier).

Résoudre le problème d'emploi du temps revient à associer à chaque activité l'un des ses créneaux possibles. Ce qui peut se résumer à affecter un tableau d'entiers non signés : chaque activité est représentée par l'indice du tableau et la valeur associé à chaque élément du tableau représente le numéro du créneau affecté.

Par exemple :

- `data[0] = 0`
- `data[1] = 2`
- `data[2] = 1`

représente, pour l'exemple décrit, l'emploi du temps suivant :

- « CM de Programmation impérative (novice) », le 9 octobre de 9h45 à 11h15 dans l'amphi 300
- « TP1a de Programmation impérative (novice) », le 9 octobre de 8h00 à 11h00 dans la salle PAS136
- « déjeuner lundi Groupe 1a », le 9 octobre de 12h00 à 13h30 (créneau sans salle)

Il va sans dire que cet emploi du temps viole quelques contraintes…

2.3 Contraintes

Le type des contraintes faibles devra être soit :

- **`precedence`** signifiant qu'une activité doit être programmée avant une autre ;
- **`closeness`** signifiant qu'une activité doit être programmée au plus près d'une autre ;
- **`occupancy`** signifiant que chaque salle d'un groupe de salle doit voir ses créneaux le plus possible regroupés au sein d'une même journée ;
- **`efficiency`** signifiant que chaque membre d'un groupe de personnes doit avoir le moins de « trous » au sein d'une même journée ;
- **`dispatch`** signifiant que chaque membre d'un groupe de personnes doit avoir une répartition journalière la plus stable (chaque journée devra comporter autant que faire se peut le même nombre de minutes de programmation horaire).

L'analyse du fichier *YAML* peut se faire au moyen d'une bibliothèque dédiée ou par l'intermédiaire des fonctions standards de lecture formattée du langage C.

2.4 Résolution

La résolution d'un problème d'affectation par algorithme génétique passe par l'écriture d'une fonction d'évaluation retournant un nombre entier d'autant plus grand qu'aucune contrainte n'est violée.

Dans le cas d'un problème d'emploi du temps, nous avons :

- une contrainte forte pour tout couple d'activités contenant la même personne (cette personne ne peut être programmée au même moment pour ces deux activités)
- une contrainte forte pour tout couple d'activités contenant dans ses créneaux possibles une même salle au même moment (cette salle ne peut être utilisée au même moment pour ces deux activités)

Et nous avons aussi les contraintes faibles exprimées dans le fichier du problème.

À chaque type de contrainte (**`strong`**, **`precedence`**, **`closeness`**, **`occupancy`**, **`efficiency`**, **`dispatch`**) est affecté une valeur. Si une affectation de créneaux spatio-temporels à l'ensemble des activités ne viole aucune contrainte, alors son évaluation est égale à la somme des valeurs des contraintes non violées. Pour chaque contrainte violée, la valeur d'une affectation est réduite de la valeur de cette contrainte.

Par défaut, les valeurs associées à chaque contrainte sont :

- **`strong`** 100
- **`precedence`** 10
- **`closeness`** 5
- **`occupancy`** 2
- **`efficiency`** 3
- **`dispatch`** 3

2.5 Exécutable

L'exécutable à écrire devra s'appeler **`timetable`** et devra prendre les arguments obligatoires suivants :

- **`--problem`** la description du problème

Il pourra avoir aussi les arguments suivants :

- **`--constraint-strong`** (valeur des contraintes fortes non violées) ;
- **`--constraint-precedence`** (valeur des contraintes de précédence non violées) ;
- **`--constraint-closeness`** (valeur des contraintes de proximité non violées) ;
- **`--constraint-occupancy`** (valeur des contraintes fortes non violées) ;
- **`--constraint-efficiency`** (valeur des contraintes d'occupation non violées) ;

- **`--constraint-dispatch`** (valeur des contraintes de répartition non violées) ;
- **`--mutation`** (probabilité de mutation ; 0.01 par défaut) ;
- **`--crossover`** (probabilité de croisement ; 0.8 par défaut) ;
- **`--population`** (nombre d'individus par population ; 100 par défaut) ;
- **`--iteration`** (nombre de populations générées ; 100 par défaut) ;
- **`--output`** la production de l'emploi du temps au format *iCal* (si le paramètre de sortie n'est pas présent, l'exécutable produira son affichage à l'écran).

```
$ ./timetable --problem myproblem.yaml --output mycalendar.ics
```

L'exécutable pourra afficher la moyenne des évaluation de chaque population pour montrer son évolution au cours du temps.

Vous pourrez vous inspirer du fichier `solve.c` qui résoud une équation du troisième ordre à l'aide d'algorithme génétique.

2.6 Données

Vous essaierez de modéliser les contraintes d'emploi du temps de la licence informatique pour la semaine du 18 au 24 septembre de la L1 à la L3.

2.7 Rapport

Le rapport est à produire au moyen de la commande

```
$ make docs
```

et devra contenir votre approche ainsi que vos difficultés et le cas échéant, ce qui ne fonctionne pas. Il devra aussi contenir, si le programme fonctionne jusqu'au bout un exemple d'emploi du temps généré au format *iCal* ainsi que la courbe d'évolution de la valeur moyenne de chaque population. Vous pourrez aussi tester plusieurs valeurs des paramètres liés aux contraintes ou à l'algorithme.

Historique des modifications

2023-2024_1 *Mercredi 11 octobre 2023*

Dr Christophe Demko <christophe.demko@univ-lr.fr>

- Version initiale

2023-2024_2 *Vendredi 10 novembre 2023*

Dr Christophe Demko <christophe.demko@univ-lr.fr>

- Ajout des instructions pour la présence d'un fichier `README.md` dans le rendu.

