ENSTA Paris
2023-2024

Résolution de problèmes algorithmiques – IN103

Alexandre Chapoutot

Feuille d'exercices 6

Objectif(s)

- Modélisation de problèmes de décision avec des (di)graphes ;
- Mise en œuvre des algorithmes de plus court chemin, arbre couvrant de poids minimal.

PRÉPARATION

Cette première partie permet de préparer votre environnement (**si vous ne l'avez pas fait au dernier TD**) de travail afin de pouvoir utiliser facilement la bibliothèque logicielle **libin103** spécialement développée pour cet enseignement.

1. A la racine de votre compte, créez un répertoire nommé `Library` s'il n'a pas déjà été créé, puis placez vous dans ce répertoire.

   ```
   mkdir ~/Library; cd ~/Library
   ```

2. Téléchargez l'archive `libin103-1.4.tar.gz` sur le site du cours

   ```
   wget https://perso.ensta-paris.fr/~chapoutot/teaching/in103/practical-work/libin103-1.4.tar.gz
   ```

3. Désarchivez l'archive

   ```
   tar -xvzf libin103-1.4.tar.gz
   ```

4. Allez dans le répertoire `libin103-1.4` et compilez la bibliothèque. Quelle commande faut-il utiliser ?
   — Il faut utiliser la commande `make`. A la fin de la compilation vérifier la présence du fichier `libin103.a` dans le répertoire `source`.
   — Également, vous pouvez exécuter la commande `make check` pour compiler et exécuter les programmes de tests.

Matériel pour le TP

Récupérez l'archive associé à cette séance de TP à l'adresse :

```
https://perso.ensta-paris.fr/~chapoutot/teaching/in103/practical-work/in103-td6.tar.gz
```

Exercice 1 – Problèmes de pavés

Fonctions utiles pour cet exercice

- `integer_graph_init`
- `integer_graph_destroy`
- `integer_graph_ins_vertex`
- `integer_graph_ins_edge`
- `integer_graph_ins_edge`
- `integer_mst`
- `generic_list_head`
- `generic_list_next`
- `generic_list_data`

Accessibles dans les fichiers en-tête : `graph.h`, `graphalg.h`, `list.h`

Il était une fois une ville qui n'avait pas de rues[1]. Il était très difficile de circuler dans la ville après de fortes pluies car le sol était boueux, les voitures s'embourbaient et les bottes des habitants étaient toutes crottées. Le maire de la ville décida de paver certaines rues mais il ne voulait pas dépenser plus que nécessaire car il voulait également faire construire une bibliothèque pour la ville. Le maire spécifia donc deux conditions :

1. Paver suffisamment de rues pour que chacun des habitants puisse se rendre de sa maison à n'importe quelle autre maison en empruntant des rues pavées.

2. Dépenser le moins d'argent possible pour paver ces rues.

L'agencement de la ville est représenté à la figure 1. Le nombre de pavés entre chaque maison représente la dépense à engager pour paver la route. Trouvez le meilleur chemin pour relier toutes les maisons mais en utilisant le moins de jetons (pavés) possible.

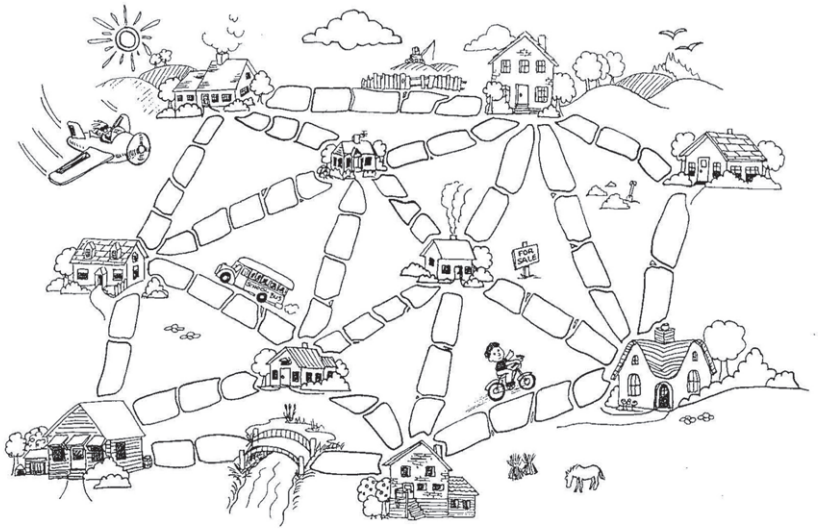
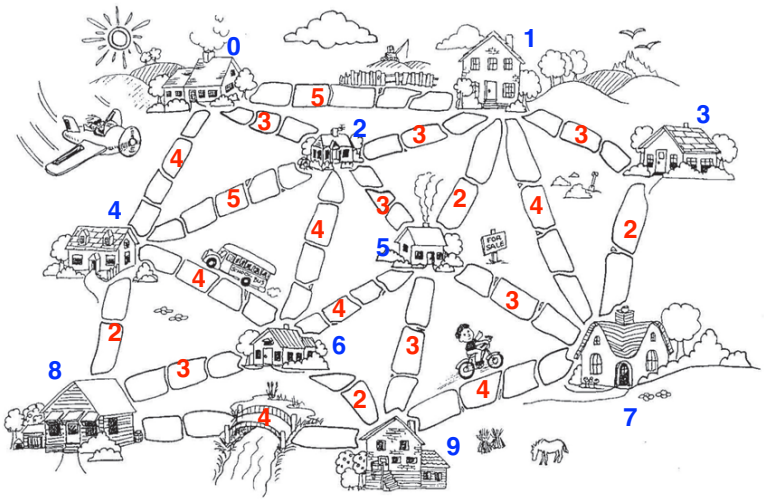


FIGURE 1 – Plan de la ville

Question 1

Modélisez l'organisation de la ville par un graphe pondéré.

Solution:



Explication(s):

- ✿ Les maisons représentent les sommets du graphes. Il faut numéroter ces maisons (nombre en bleu dans l'image).
- ✿ Les routes représentent les arcs du graphes.
- ✿ Le nombre de pavés composant les chemins entre les maisons représentent les pondérations des arcs (nombre en rouge dans l'image).

Question 2

Utilisez l'algorithme adéquat et présent dans la bibliothèque `libin103` pour proposer une solution au maire de la ville. Aidez-vous du squelette de programme donné dans le répertoire `exo1`, pour résoudre informatiquement ce problème.

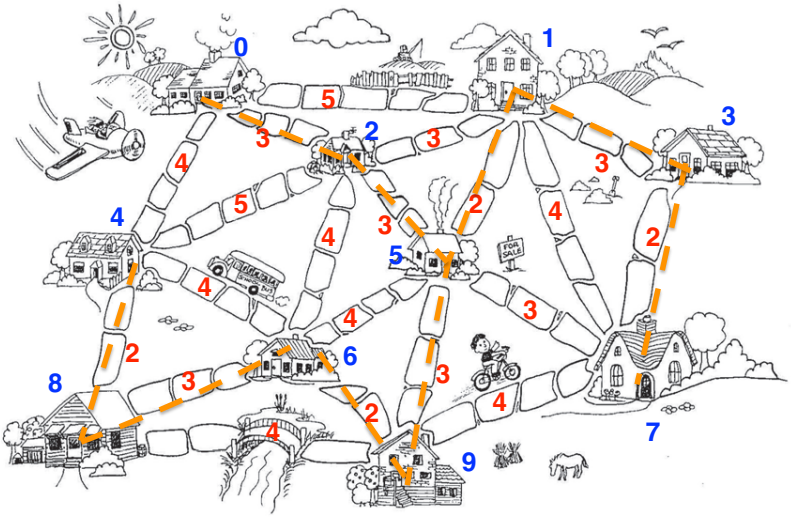
Solution:

Explication(s):

- ✿ La solution de ce problème est donnée en appliquant un algorithme de calcul de l'arbre couvrant de poids minimal. La fonction `integer_mst` met en œuvre l'algorithme de Kruskal pour cela.
- ✿ Pas beaucoup de difficultés dans le code de la solution
  - — Essentiellement, il faut penser à insérer deux fois les arcs, dans un sens puis dans l'autre pour bien modéliser un graphe non orienté.
  - — Autre point de vigilance, l'initialisation de la liste représentant l'arbre couvrant est réalisée dans la fonction `integer_mst`. Essentiellement, car nous manipulons une liste générique (`void*`) qui nécessite une fonction de construction des éléments de la liste qui est donnée dans le fichier contenant la fonction `integer_mst`.
  - — L'affichage de la liste d'élément de l'arbre couvrant nécessite de « transtyper » le résultat de la fonction `generic_list_data` pour accéder aux données décrites par la structure `we_t`.

```
int main (int argc, char** argv) {
  integer_graph_t graph;
  integer_graph_init (&graph);

  for (int i = 0; i < 10 ; i++) {
    integer_graph_ins_vertex(&graph, i);
  }
  /* The graph is not directed so we enter the edges in one direction
     and in the other direction */
  integer_graph_ins_edge(&graph, 0, 1, 5.0); integer_graph_ins_edge(&graph, 1, 0, 5.0);
  integer_graph_ins_edge(&graph, 0, 2, 3.0); integer_graph_ins_edge(&graph, 2, 0, 3.0);
  integer_graph_ins_edge(&graph, 0, 4, 4.0); integer_graph_ins_edge(&graph, 4, 0, 4.0);

  integer_graph_ins_edge(&graph, 1, 2, 3.0); integer_graph_ins_edge(&graph, 2, 1, 3.0);
  integer_graph_ins_edge(&graph, 1, 3, 3.0); integer_graph_ins_edge(&graph, 3, 1, 3.0);
  integer_graph_ins_edge(&graph, 1, 5, 2.0); integer_graph_ins_edge(&graph, 5, 1, 2.0);
  integer_graph_ins_edge(&graph, 1, 7, 4.0); integer_graph_ins_edge(&graph, 7, 1, 4.0);

  integer_graph_ins_edge(&graph, 2, 4, 5.0); integer_graph_ins_edge(&graph, 4, 2, 5.0);
  integer_graph_ins_edge(&graph, 2, 5, 3.0); integer_graph_ins_edge(&graph, 5, 2, 3.0);
  integer_graph_ins_edge(&graph, 2, 6, 4.0); integer_graph_ins_edge(&graph, 6, 2, 4.0);

  integer_graph_ins_edge(&graph, 3, 7, 2.0); integer_graph_ins_edge(&graph, 7, 3, 2.0);

  integer_graph_ins_edge(&graph, 4, 6, 4.0); integer_graph_ins_edge(&graph, 6, 4, 4.0);
  integer_graph_ins_edge(&graph, 4, 8, 2.0); integer_graph_ins_edge(&graph, 8, 4, 2.0);

  integer_graph_ins_edge(&graph, 5, 6, 4.0); integer_graph_ins_edge(&graph, 6, 5, 4.0);
  integer_graph_ins_edge(&graph, 5, 7, 3.0); integer_graph_ins_edge(&graph, 7, 5, 3.0);
  integer_graph_ins_edge(&graph, 5, 9, 3.0); integer_graph_ins_edge(&graph, 9, 5, 3.0);

  integer_graph_ins_edge(&graph, 6, 8, 3.0); integer_graph_ins_edge(&graph, 8, 6, 3.0);
  integer_graph_ins_edge(&graph, 6, 9, 2.0); integer_graph_ins_edge(&graph, 9, 6, 2.0);

  integer_graph_ins_edge(&graph, 7, 9, 4.0); integer_graph_ins_edge(&graph, 9, 7, 4.0);

  integer_graph_ins_edge(&graph, 8, 9, 4.0); integer_graph_ins_edge(&graph, 9, 8, 4.0);

  print_graph (&graph);
  printf ("\n\n");

  generic_list_t *span;
  integer_mst (&graph, &span);
  generic_list_elmt_t *elem = generic_list_head (span);
  for (; elem != NULL; elem = generic_list_next (elem)) {
    we_t* wEdge = generic_list_data (elem);
    printf ("[ %d -- (%.1f) -- %d ], ", wEdge->source, wEdge->weight, wEdge->destination);
  }
  printf ("\n");

  generic_list_destroy (span);
  integer_graph_destroy (&graph);
  return EXIT_SUCCESS;
}
```

Solution:
Une version graphique de la solution



Exercice 2 – Déplacement autonome d'un robot

Fonctions utiles pour cet exercice

- `integer_graph_init`
- `integer_graph_destroy`
- `integer_graph_ins_vertex`
- `integer_graph_ins_edge`
- `integer_graph_ins_edge`
- `integer_shortest`
- `generic_list_head`
- `generic_list_next`
- `generic_list_data`

Accessibles dans les fichiers en-tête : `graph.h`, `graphalg.h`, `list.h`

On considère un jeu vidéo représenté par une carte décrite par une grille qui représente l'environnement dans lequel doit se déplacer un joueur. Une case de cette grille peut être de différents types en fonction des éléments de la nature quelle représente. Nous considérons différents éléments naturels ou artificiels auxquels nous attribuons un coût de traversabilité (décrivant un effort plus ou moins important pour traverser un type de terrain) qui sont décrits dans la figure 2.

Pour cet exercice, nous considérerons la carte décrite par la grille donnée à la figure 3. Les déplacements du joueur ne font que par les faces (nord, sud, est, ouest) des cases de la grille. Le joueur doit aller du point vert au point jaune pour réaliser sa mission en produisant le minimum d'effort.

Types de terrain	Illustration	Coût de traversabilité
Abîme[2]		∞
Rivière[3]		∞
Route[4]		1
Pont[5]		1
Prairie[6]		2

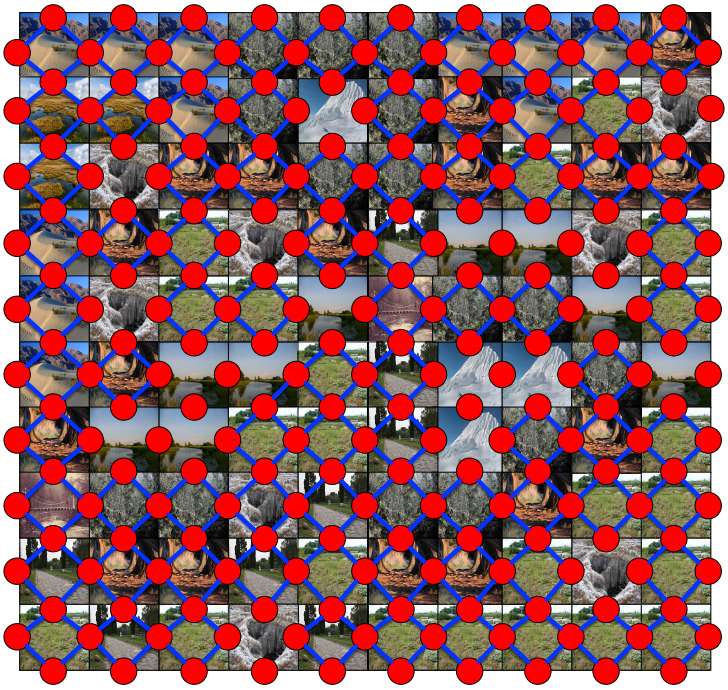
Types de terrain	Illustration	Coût de traversabilité
Chemin forestier[7]		3
Désert[8]		4
Forêt dense[9]		4
Marécage[10]		5
Montagne[11]		∞

FIGURE 2 – Types de terrain

Question 1

Donnez une modélisation à l'aide d'un graphe orienté pondéré de cette carte.

Solution:



Explication(s):

- ✿ Comme indiqué dans l'énoncé, les déplacements se font que par les faces des cases de la grille.
- ✿ On associe un sommet du graphe à chaque centre de chaque côté de chaque case (les points rouges dans l'image).
- ✿ On ajoute un arcs entre chaque sommet (dans les deux sens, pas représenté sur l'image pour ne pas alourdir celle-ci). Les poids des arcs dépendent de l'image sous-jacente selon la nomenclature donnée dans l'énoncé.
- ✿ On n'ajoute pas les arcs entre les sommets quand l'image représente un obstacle infranchissable (c'est-à-dire, une montage, une rivière ou un abîme).

Question 2

Le nombre de sommets du graphe étant conséquent pour une simple grille de dimension 10×10, on va automatiser

FIGURE 3 – Grille de jeu

un petit peu la modélisation. Pour cela on va lire un fichier formé :

- d'une première ligne qui contient un entier positif donnant la dimension de la grille ;
- d'une matrice d'entiers positifs décrivant la carte où chaque cœfficient de la matrice représente un type de terrain suivant la convention :
  - Abîme = 0
  - Rivière = 1
  - Route = 2
  - Pont = 3
  - Prairie = 4
  - Chemin forestier = 5
  - Désert = 6
  - Forêt dense = 7
  - Marécage = 8
  - Montagne = 9

  Par exemple, la carte de la figure 3 sera représentée par le fichier `carte.txt` dont le contenu est

```
10
6667776665
8867975640
8055775455
6540521104
6044137714
6511429971
5114429754
3770277544
2552455404
4244244444
```

En conséquence, il faut trouver une numérotation adéquate pour représenter les sommets du graphe par rapport aux coordonnées de la matrice (ligne/colonne). Pour cela on considérera que les sommets sont sur une grille deux fois plus fines que le systèmes de coordonnées de la matrice.

Donnez la règle qui transforme une coordonnées d'un cœfficient de la matrice dans ce nouveau système de coordonnées.

Solution:

$$(x,y) \mapsto (x',y') = (2x+1, 2y+1)$$

Question 3

Pour une case de coordonnées (x',y') de la nouvelle grille, donnez les coordonnées des centres de ces faces (nord, sud, est, ouest).

Solution:

$$\begin{aligned} \text{nord}(x,y) &= (x, y-1) \\ \text{sud}(x,y) &= (x, y+1) \\ \text{est}(x,y) &= (x+1, y) \\ \text{ouest}(x,y) &= (x-1, y) \end{aligned}$$

Question 4

Maintenant, nous voulons une notation par un entier et non par un couple d'entiers afin de pouvoir utiliser la représentation des graphes fournie par la bibliothèque `libin103`. Quelle transformation permet de faire cela ?

Indice : pour cette dernière transformation, pensez à la représentation d'une matrice par un tableau unidimensionnel.

Solution:

$$(x',y') \mapsto (x'',y'') = x' \times 2 \times \text{dimension} + y'$$

Explication(s):

- ✿ Note : (x',y') représente les coordonnées d'une case dans le système de coordonnées le plus fin d'où le 2 fois la dimension de la grille initiale.

Question 5

Dans le squelette de programme donné dans le répertoire `exo2`, codez le corps de la fonction `buildGraph` qui permet d'ajouter les sommets et les arcs du graphe modélisant la carte.

Solution:

```
int buildGraph (integer_graph_t *graph, char* filename) {
  FILE* fd = fopen (filename, "r");
  if (fd == NULL) {
    fprintf (stderr, "Ne peux pas ouvrir le fichier %s\n", filename);
    return 0;
  }

  int dimension;
  fscanf (fd, "%d", &dimension);
  printf ("Dimension de la carte est %d\n", dimension);

  for (int i = 0; i < dimension; i++) {
    for (int j = 0; j < dimension+1; j++) {
      char c  = fgetc(fd);
      if (! isspace(c)) {       /* Note: le saut de ligne est en debut */
        double cout = traversabilite ( (int) (c - 48));
        int nord = (2 * i) * 2 * dimension + 2 * (j-1) + 1;
        int sud = (2 * i + 2) * 2 * dimension + 2 * (j-1) + 1;
        int est = (2 * i + 1) * 2 * dimension + 2 * (j-1) + 2;
        int ouest = (2 * i + 1) * 2 * dimension + 2 * (j-1);

        integer_graph_ins_vertex (graph, nord);
        integer_graph_ins_vertex (graph, sud);
        integer_graph_ins_vertex (graph, est);
        integer_graph_ins_vertex (graph, ouest);

        if(cout < 1e308) {
          integer_graph_ins_edge (graph, nord, est, cout);
          integer_graph_ins_edge (graph, est, nord, cout);
          integer_graph_ins_edge (graph, nord, ouest, cout);
          integer_graph_ins_edge (graph, ouest, nord, cout);
          integer_graph_ins_edge (graph, sud, est, cout);
          integer_graph_ins_edge (graph, est, sud, cout);
          integer_graph_ins_edge (graph, sud, ouest, cout);
          integer_graph_ins_edge (graph, ouest, sud, cout);
        }
      }
    }
  }

  fclose(fd);
  return dimension;
}
```

Explication(s):

- Plusieurs points de vigilance :
  - La conversion d'un caractère en entier peut se faire simplement par un décalage par la valeur du code ASCII du caractère 0 (la valeur 48), cf ligne 16 du bout de code donnée en solution.
  - La lecture des lignes du fichier `carte.txt` induit un décalage dans l'indice j à cause du saut de ligne. Il faut donc décaler de 1 la variable j pour bien avoir des chiffres en 0 et 9.
- Une fois calculer les valeurs des entiers associés aux faces nord, sud, est, ouest d'une case de la carte, il suffit d'ajouter les sommets au graphes.
- L'ajout des arcs dépend du type de terrain mais il ne faut pas oublier d'ajouter les arcs dans les deux sens.
- On représente une valeur infinie dans le graphe par la valeur flottante $1e308$.

Question 6

Utilisez l'algorithme adéquat et présent dans la bibliothèque `libin103` pour trouver le chemin demandant le moins d'effort pour le joueur. Aidez-vous du squelette de programme donné dans le répertoire `exo2`, pour résoudre informatiquement ce problème.

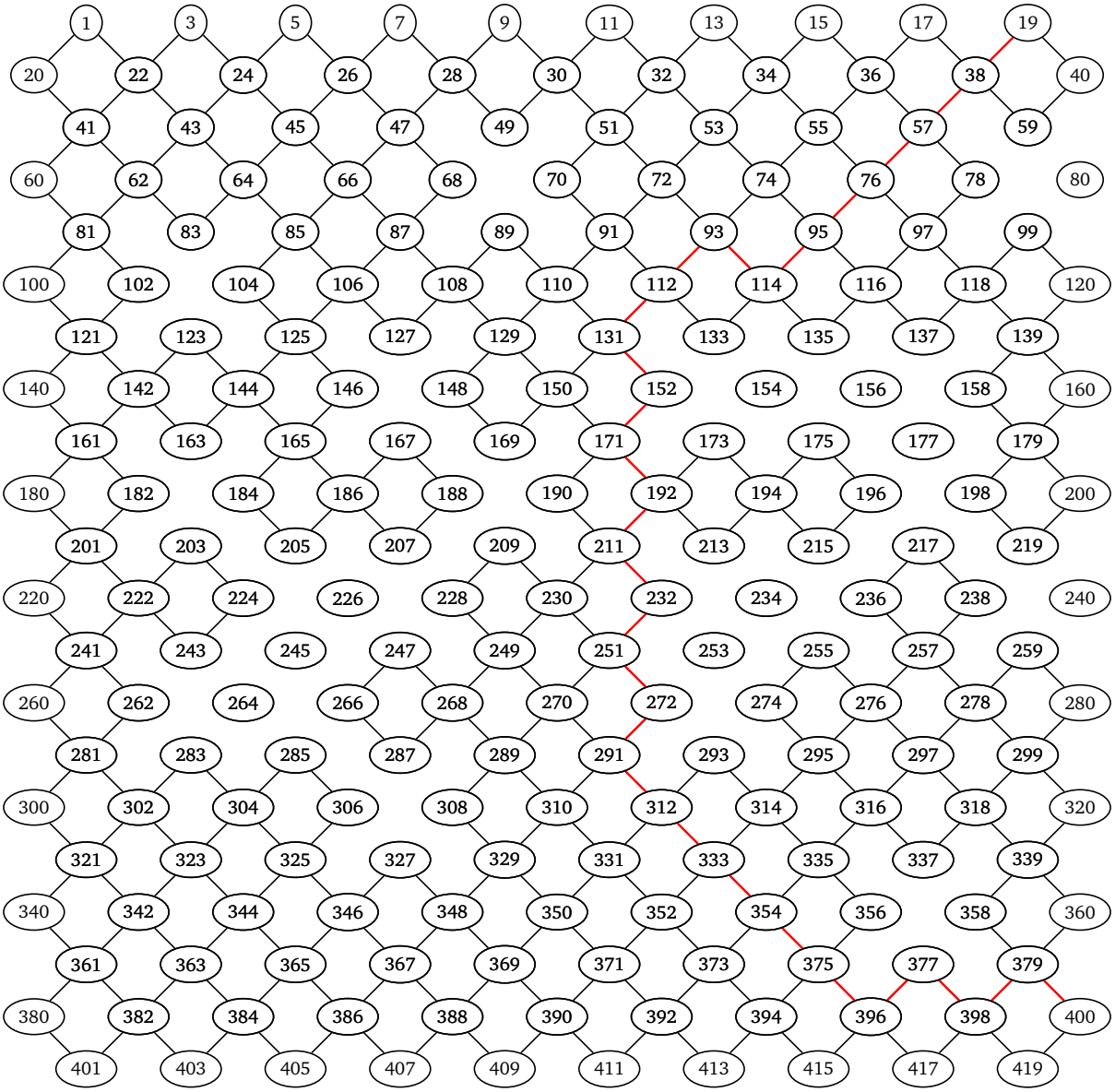
Solution:

```
integer_graph_t graph;
integer_graph_init (&graph);

int dimension = buildGraph (&graph, argv[1]);

print_graph (&graph);

/* Sommet a l'est de la derniere case de la derniere ligne */
int start = (2 * 9 + 1) * 2 * dimension + (2 * 9 + 2);
/* Sommet au nord de la derniere case de la premiere ligne */
int end = (0 * 9) * 2 * dimension + (2 * 9 + 1);

generic_list_t *paths;
integer_shortest (&graph, start, &paths);

/* On recherche le chemin en transformant le resultat en tableau de parents */
int* parents = malloc ((2*dimension*2*dimension)*sizeof(int));
if (parents == NULL) {
  generic_list_destroy (paths);
  integer_graph_destroy (&graph);
  return EXIT_FAILURE;
}

generic_list_elmt_t *elem = generic_list_head (paths);
for (; elem != NULL; elem = generic_list_next (elem)) {
  ed_t* v = generic_list_data (elem);
  parents[v->vertex] = v->parent;
}

int u = end;
while (u != -1) {
  printf ("%d <- ", u);
  u = parents[u];
}
printf ("\n");

generic_list_destroy (paths);
integer_graph_destroy (&graph);
```

Explication(s):

- Essentiellement une fois le problème modélisé par un graphe pondéré, il suffit d'utiliser l'algorithme de Dijkstra pour calculer le chemin de poids minimum (la fonction `integer_shortest`.
- Cependant, le résultat est une liste chaîné de l'arbre de tous les plus courts chemins depuis le sommet de départ choisi. Pour calculer rapidement le chemin entre le départ et l'arrivée, nous convertissons cette liste chaînée en un tableau qui lie un sommet avec son parent. Cela permet de faire une recherche arrière (depuis la fin) plus rapidement (cf lignes 17 à 35).
- Note une difficulté du codage utilisé pour numéroter les sommets est qu'il n'est pas injectif donc il est difficile de retourner aux coordonnées de départ pour bien visualiser le résultat (cf la figure de la correction pour avoir une idée du déplacement).

Solution:
La solution du problème est donnée par le chemin suivant



APPROFONDISSEMENT

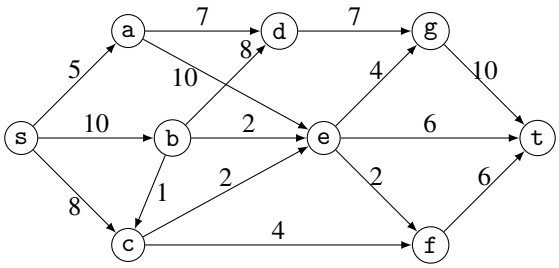
Exercice 1 – Problème de logistique

Fonctions utiles pour cet exercice

- `integer_graph_init`
- `integer_graph_destroy`
- `integer_graph_ins_vertex`
- `integer_graph_ins_edge`
- `integer_graph_ins_edge`
- `integer_maxflow`
- `generic_list_head`
- `generic_list_next`
- `generic_list_data`

Accessibles dans les fichiers en-tête : `graph.h`, `graphalg.h`, `list.h`

On considère un problème de circulation[12] de cartons de bonbons entre différentes villes. Le réseau de transit est décrit par le digraphe pondéré suivant



Les poids représentent les tonnes de cartons de bonbons véhiculés entre chaque ville.

La personne en charge du réseau de distribution indique à sa hiérarchie que le réseau n'est pas capable de véhiculer plus 15 tonnes de cartons bonbons.

Question 1

Écrivez un programme s'appuyant sur la bibliothèque `libin103` et le squelette de code situé dans le répertoire `exo3` pour vérifier cette affirmation.

Solution:
Le flot maximal est de 20 tonnes de cartons de bonbons et non 15 tonnes.

```c
int main (int argc, char** argv) {
  integer_graph_t graph;
  integer_graph_init (&graph);

  for (int i = 0; i < 9 ; i++) {
    integer_graph_ins_vertex(&graph, i);
  }
  integer_graph_ins_edge(&graph, 0, 1, 5.0);
  integer_graph_ins_edge(&graph, 0, 2, 10.0);
  integer_graph_ins_edge(&graph, 0, 3, 8.0);

  integer_graph_ins_edge(&graph, 1, 4, 7.0);
  integer_graph_ins_edge(&graph, 1, 5, 10.0);

  integer_graph_ins_edge(&graph, 2, 3, 1.0);
  integer_graph_ins_edge(&graph, 2, 4, 8.0);
  integer_graph_ins_edge(&graph, 2, 5, 2.0);

  integer_graph_ins_edge(&graph, 3, 5, 2.0);
  integer_graph_ins_edge(&graph, 3, 6, 4.0);

  integer_graph_ins_edge(&graph, 4, 7, 7.0);

  integer_graph_ins_edge(&graph, 5, 6, 2.0);
  integer_graph_ins_edge(&graph, 5, 7, 4.0);
  integer_graph_ins_edge(&graph, 5, 8, 6.0);

  integer_graph_ins_edge(&graph, 6, 8, 6.0);

  integer_graph_ins_edge(&graph, 7, 8, 10.0);

  print_graph (&graph);
  printf ("\n\n");

  double** localflows;
  double flow;
  integer_maxflow (&graph, 0, 8, &localflows, &flow);
  for (int i = 0; i < 9; i++) {
    for (int j = 0; j < 9; j++) {
      if (i != j && localflows[i][j] > 0.0) {
        printf ("%d -(%.1f)-> %d; ", i, localflows[i][j], j);
      }
    }
    printf ("\n");
  }
  printf ("flow=%f\n", flow);

  integer_graph_destroy (&graph);
  return EXIT_SUCCESS;
}
```

Explication(s):

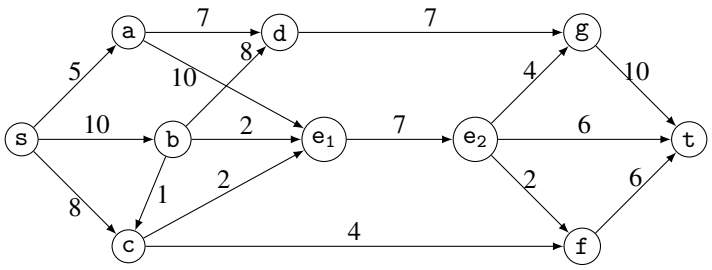
- La numérotation des sommets dans le programme C suit la convention suivante :
  — $s = 0$, $a = 1$, $b = 2$, $c = 3$, $d = 4$, $e = 5$, $f = 6$, $g = 7$ et $t = 8$.
- On a ici un problème de flot maximal. Nous utilisons donc après une modélisation informatique de ce graphe, nous utilisons la fonction `integer_maxflow` pour calculer la valeur du flot maximal.
- Attention l'allocation mémoire de la matrice `localflows` est réalisée dans la fonction `integer_maxflow`.
- Par défaut la fonction `integer_maxflow` affiche tous les chemins améliorant trouver lors du déroulement de l'algorithme de Edmonds-Karp.
- Le parcours de la matrice `localflow` permet de retrouver les chemins du flot maximal.

Question 2

Pendant une courte période, la ville e va effectuer des travaux de voirie et en conséquence seulement 7 tonnes de cartons de bonbons pourront transités par la ville. Comment modéliser cette contrainte dans le graphe de distribution ? Quel impact cela a sur le volume total transportable ?

Solution:

La modélisation par graphe d'un problème de flot ne prend en compte que les contraintes de capacité sur les arcs par sur les sommets. Pour prendre en compte cette contrainte sur un sommet nous allons dédoubler ce sommet e en deux sommets e_1 et e_2 et relier ces deux sommets par un arcs de capacité 7. Au final on a le graphe



Solution:
Pour le nouveau code mettant en œuvre cet algorithme on fait des changements à minima.

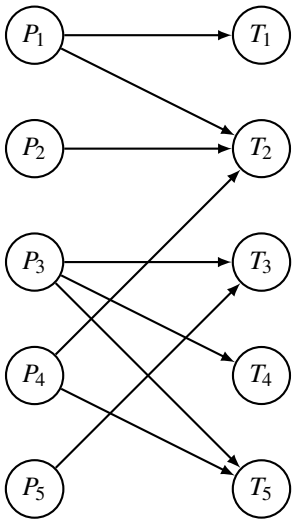
```
int main (int argc, char** argv) {
  integer_graph_t graph;
  integer_graph_init (&graph);

  for (int i = 0; i < 10 ; i++) {
    integer_graph_ins_vertex(&graph, i);
  }
  integer_graph_ins_edge(&graph, 0, 1, 5.0);
  integer_graph_ins_edge(&graph, 0, 2, 10.0);
  integer_graph_ins_edge(&graph, 0, 3, 8.0);

  integer_graph_ins_edge(&graph, 1, 4, 7.0);
  integer_graph_ins_edge(&graph, 1, 5, 10.0);

  integer_graph_ins_edge(&graph, 2, 3, 1.0);
  integer_graph_ins_edge(&graph, 2, 4, 8.0);
  integer_graph_ins_edge(&graph, 2, 5, 2.0);

  integer_graph_ins_edge(&graph, 3, 5, 2.0);
  integer_graph_ins_edge(&graph, 3, 6, 4.0);

  integer_graph_ins_edge(&graph, 4, 7, 7.0);

  /* Nouvel arc */
  integer_graph_ins_edge(&graph, 5, 9, 7.0);

  integer_graph_ins_edge(&graph, 9, 6, 2.0);
  integer_graph_ins_edge(&graph, 9, 7, 4.0);
  integer_graph_ins_edge(&graph, 9, 8, 6.0);

  integer_graph_ins_edge(&graph, 6, 8, 6.0);

  integer_graph_ins_edge(&graph, 7, 8, 10.0);

  print_graph (&graph);
  printf ("\n\n");

  double** localflows;
  double flow;
  integer_maxflow (&graph, 0, 8, &localflows, &flow);
  for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 10; j++) {
      if (i != j && localflows[i][j] > 0.0) {
        printf ("%d -(%.1f)-> %d; ", i, localflows[i][j], j);
      }
    }
    printf ("\n");
  }
  printf ("flow=%f\n", flow);

  integer_graph_destroy (&graph);
  return EXIT_SUCCESS;
}
```

Explication(s):

- On ajoute un nouveau sommet noté 9 pour représenter le sommet e_2 (ligne 5). Le sommet e est toujours numéroté 5.
- On ajoute un nouvel arc entre les sommets 5 et 9 avec une capacité de 7 (ligne 25)
- On change les limites d'affichage de la matrice (lignes 41 et 42)
- Le nouveau flot maximum est de 15.

Exercice 2 – Compétence vs tâches

On considère un problème d'allocation de 5 tâches à 5 personnes suivant leurs compétences. Pour cela, on considère un graphe bipartite de la forme suivante :



Une personne est associée à une tâche dans le graphe bipartite quand cette personne a la compétence pour réaliser cette tâche. Les personnes sont représentées par P_i avec $1 \leqslant i \leqslant 5$ et les tâches sont représentées par T_i avec $1 \leqslant i \leqslant 5$.

On peut se ramener à un problème de calcul de flot maximum en ajoutant deux sommets s et t tel que s est connecté à tous les sommets P_i et tous les sommets T_i sont connecté au sommet t. Tous les arcs du graphe ont une capacité de 1 et orientés de s vers t.

Question 1

Mettez en œuvre le calcul du flot maximal pour ce problème à l'aide de la bibliothèque `libin103` et du squelette de code donné dans le répertoire `exo4`.

On prendra comme convention de numération des sommets :

— $s = 0$, $P_1 = 1$, $P_2 = 2$, …, $P_5 = 6$, $T_1 = 7$, $T_2 = 8$, …, $T_5 = 10$ et $t = 11$.

Solution:

```c
int main (int argc, char** argv) {
  integer_graph_t graph;
  integer_graph_init (&graph);

  for (int i = 0; i < 12 ; i++) {
    integer_graph_ins_vertex(&graph, i);
  }
  integer_graph_ins_edge(&graph, 1, 6, 1.0);
  integer_graph_ins_edge(&graph, 1, 7, 1.0);

  integer_graph_ins_edge(&graph, 2, 7, 1.0);

  integer_graph_ins_edge(&graph, 3, 8, 1.0);
  integer_graph_ins_edge(&graph, 3, 9, 1.0);
  integer_graph_ins_edge(&graph, 3, 10, 1.0);

  integer_graph_ins_edge(&graph, 4, 7, 1.0);
  integer_graph_ins_edge(&graph, 4, 10, 1.0);

  integer_graph_ins_edge(&graph, 5, 8, 1.0);

  /* Super source */
  for (int i = 1; i <= 5; i++) {
    integer_graph_ins_edge(&graph, 0, i, 1.0);
  }

  /* Super puits */
  for (int i = 6; i <= 10; i++) {
    integer_graph_ins_edge(&graph, i, 11, 1.0);
  }

  print_graph (&graph);
  printf ("\n\n");

  double** localflows;
  double flow;
  integer_maxflow (&graph, 0, 11, &localflows, &flow);
  for (int i = 0; i < 12; i++) {
    for (int j = 0; j < 12; j++) {
      if (i != j && localflows[i][j] > 0.0) {
        printf ("%d -(%.1f)-> %d; ", i, localflows[i][j], j);
      }
    }
    printf ("\n");
  }
  printf ("flow=%f\n", flow);

  integer_graph_destroy (&graph);
  return EXIT_SUCCESS;
}
```

— La mise en œuvre suit la programmation habituelle des graphes orientés et un appel à la fonction `integer_maxflow` pour calculer le flot maximum. Le flot donne le couplage maximum.

— A noter que dans le cas des graphe bipartite on perd l'optimalité du résultat. Par exemple dans cette implémentation le flot est de 4 alors qu'il devrait être de 5.

— Changez l'ordre des définitions d'arc aux lignes 13 et 14 et on obtient bien un flot de 5.

Notes

[1] Cet exercice est issu de la plateforme « Computer Science Unplugged » accessible à l'adresse `https://classic.csunplugged.org/activities/minimal-spanning-trees/`

[2] Source `https://commons.wikimedia.org/wiki/File:Fluted_pothole_-_geograph.org.uk_-_116344.jpg`

[3] Source `https://commons.wikimedia.org/wiki/File:Waver_near_Bullewijk.jpg`

[4] Source `https://commons.wikimedia.org/wiki/File:RomaViaAppiaAntica03.JPG`

[5]Source https://commons.wikimedia.org/wiki/File:Lavacherie_Pont_Suspendu.jpg
[6]Source https://commons.wikimedia.org/wiki/File:Steppe-garden.JPG
[7]Source https://commons.wikimedia.org/wiki/File:Chemin_randonnee_ile_grande.JPG
[8]Source https://commons.wikimedia.org/wiki/File:Death_valley_sand_dunes.jpg
[9]Source https://commons.wikimedia.org/wiki/File:Australian_bush02.jpg
[10]Source https://commons.wikimedia.org/wiki/File:A_peat_bog_below_the_top_of_Doune_Hill,_Luss_Hills,_Scotland.jpg
[11]Source https://commons.wikimedia.org/wiki/File:Alpamayo.jpg
[12]Exercice inspiré d'un exercice de « Exercices et problèmes résolus de recherche opérationnelle » Tome I – Roseaux – paru chez MASSON