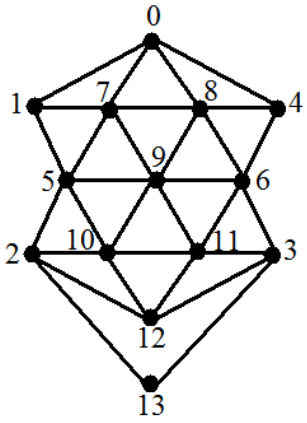
Exercices d'examen sur les graphes (niveau L3) avec corrigés

1) Exploration d'un graphe



Pour ce graphe non orienté à 14 sommets, les voisins de chaque sommet sont supposés écrits dans l'ordre **croissant** de leurs numéros. Ainsi

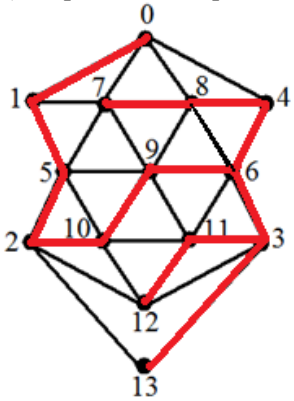
0 a pour voisins 1, 4, 7, 8 ;
1 a pour voisins 0, 5, 7 ;
2 a pour voisins 5, 10, 12, 13 ;
etc.

1) En partant du sommet 0, faire une exploration en profondeur de ce graphe, en utilisant l'ordre de voisins tel qu'il a été défini. Dessiner l'arbre obtenu.

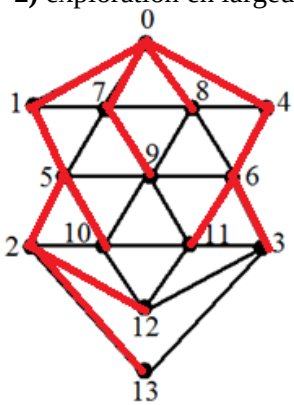
2) Toujours en partant du sommet 0, faire une exploration en largeur du graphe. On aura intérêt à utiliser l'évolution d'une file, afin de dessiner l'arbre final de l'exploration.

Corrigé :

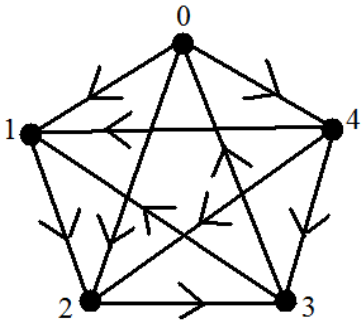
1) exploration en profondeur



2) exploration en largeur



2) Chemins hamiltoniens



Ce graphe à cinq sommets est tel que deux sommets quelconques sont reliés par un arc unique, dans un sens ou dans l'autre. Ce type de graphe a comme propriété d'avoir un nombre **impair** de chemins hamiltoniens (avec le sommet final différent du sommet initial) lorsque l'on prend les cinq points de départ possibles à tour de rôle.

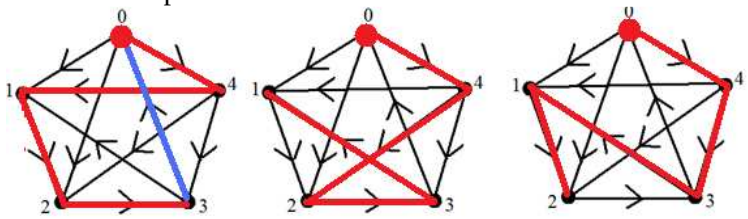
1) Déterminer tous les chemins hamiltoniens partant de chacun des 5 sommets (0, puis 1, puis 2, puis 3 et enfin 4), et se terminant en un sommet différent du point de départ. Les dessiner dans chacun de ces 5 cas. Combien y en a-t-il ?

2) Vérifier que pour chacun des points de départ, un des chemins hamiltoniens peut être prolongé pour donner un cycle hamiltonien. Les cinq cycles hamiltoniens obtenus constituent le même cycle si l'on ne tient pas compte du point de départ.

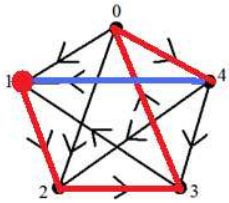
Corrigé :

1) On trouve 9 chemins hamiltoniens.

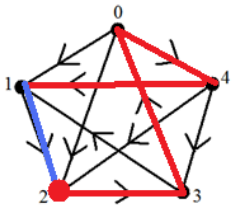
3 chemins en partant de 0



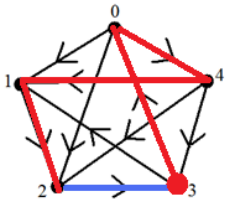
1 chemin en partant de 1



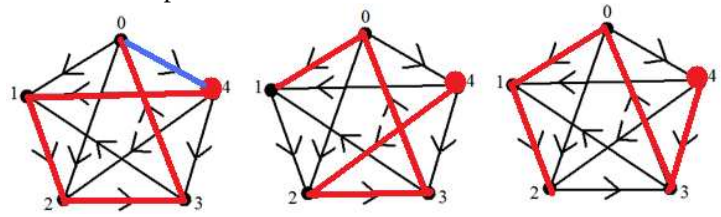
1 chemin en partant de 2



1 chemin en partant de 3

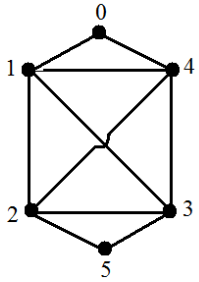


3 chemins en partant de 4



2) Pour chacun des 5 points de départ, un des chemins hamiltoniens peut être prolongé en cycle hamiltonien (arc en bleu). On trouve 5 cycles hamiltoniens, ou un seul si l'on ne tient pas compte du point de départ.

3) Cycles eulériens



Considérons ce graphe non orienté à 6 sommets numérotés de 0 à 5.

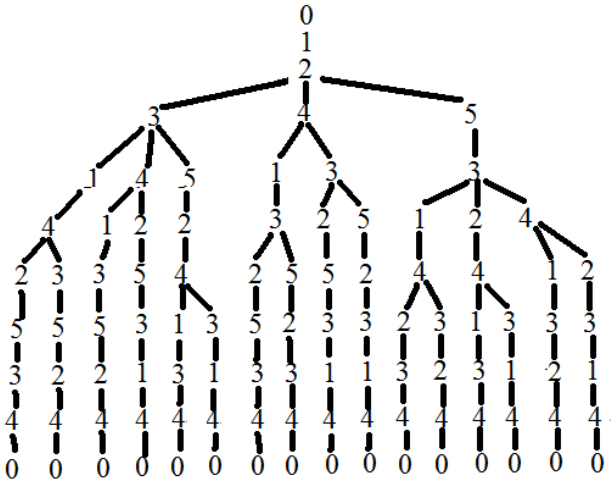
1) Pourquoi existe-t-il des cycles eulériens ?

2) Déterminer tous les cycles eulériens qui démarrent avec la succession des sommets 0 1 2.

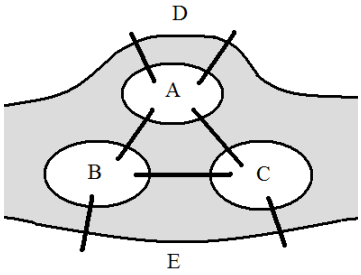
Corrigé :

1) Il existe des cycles eulériens puisque chaque sommet est de degré pair.

2) On trouve 16 cycles eulériens, construits grâce à cette arborescence (mais si on le fait à la main, mieux vaut dessiner les 16 cycles) :

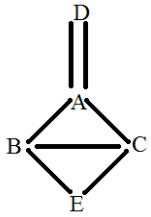


4) Chemins eulériens

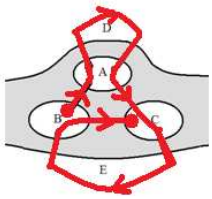


On a trois iles *A*, *B*, *C* sur une rivière dont les rives sont notées *D* et *E*, avec 7 ponts de jonction comme sur le dessin. Construire le graphe correspondant à ce schéma, et montrer qu'il possède des chemins eulériens mais pas de cycles eulériens. Autremnt dit, à condition de partir d'un endroit précis, une personne peut traverser chaque pont une fois et une seule, mais sans se retrouver finalement à son point de départ. Donner un exemple de chemin.

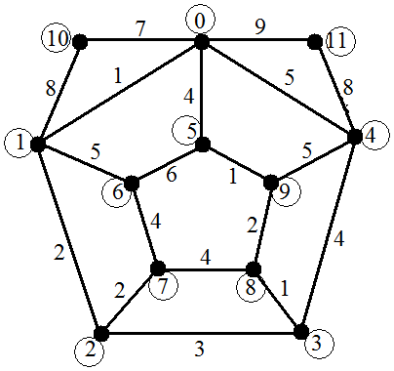
Corrigé :



Comme l'indique le graphe associé au dessin, il existe deux sommets, *B* et *C*, de degré impair, les autres étant de degré pair, d'où l'existence de chemins eulériens partant de *B* et se terminant en *C* (ou inversement). Un chemin est par exemple :



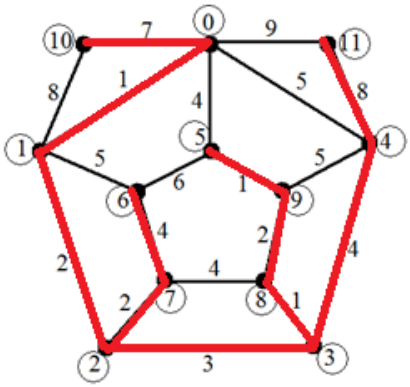
5) Arbre couvrant minimal



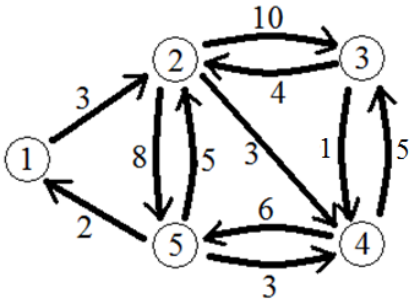
On considère ce graphe non orienté pondéré à 12 sommets (de 0 à 11), les poids étant indiqués sur les arêtes correspondantes.

En prenant comme point de départ le sommet 0, construire l'arbre couvrant minimal en appliquant l'algorithme de Prim.

Corrigé :



6) Plus courts chemins



Ce graphe orienté pondéré possède 5 sommets et 11 arcs.

1) Ecrire la matrice d'adjacence de ce graphe.

2) Utiliser l'algorithme de Floyd pour déterminer les longueurs des plus courts chemins de n'importe quel sommet vers n'importe quel autre.

Corrigé :

1)

	1	2	3	4	5
1	0	3	∞	∞	∞
2	∞	0	10	3	8
3	∞	4	0	1	∞
4	∞	∞	5	0	6
5	2	5	∞	3	0

2)

①

	1	2	3	4	5
1	0	3	∞	∞	∞
2	∞	0	10	3	8
3	∞	4	0	1	∞
4	∞	∞	5	0	6
5	2	5	∞	3	0

②

	1	2	3	4	5
1	0	3	13	6	11
2	∞	0	10	3	8
3	∞	4	0	1	12
4	∞	∞	5	0	6
5	2	5	15	3	0

③

	1	2	3	4	5
1	0	3	13	6	11
2	∞	0	10	3	8
3	∞	4	0	1	12
4	∞	9	5	0	6
5	2	5	15	3	0

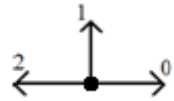
④

	1	2	3	4	5
1	0	3	11	6	11
2	∞	0	8	3	8
3	∞	4	0	1	7
4	∞	9	5	0	6
5	2	5	8	3	0

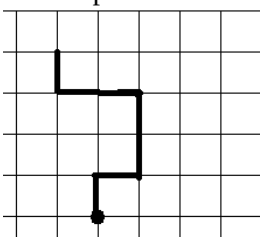
⑤

	1	2	3	4	5
1	0	3	11	6	11
2	10	0	8	3	8
3	9	4	0	1	7
4	8	9	5	0	6
5	2	5	8	3	0

7) Programmation d'un cheminement

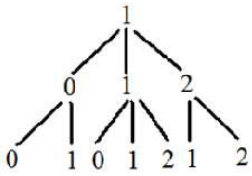
On se place dans un quadrillage de pas unité, dans lequel on dessine des chemins montants à partir d'un point du quadrillage. Puisque ces chemins sont montants, seules trois directions sont permises à chaque pas : 

. D'autre part, aucun retour n'est permis là où l'on est déjà passé. Le premier pas est toujours choisi égal à 1 (pas vertical). L'objectif est de connaître le nombre de tels chemins ayant tous une même longueur *N*, c'est-à-dire ayant *N* pas.



Un des chemins de longueur $N = 7$, soit 1011221

Cela se fait en construisant une arborescence qui commence ainsi, la lecture des chemins se faisant en descente :



Ainsi, le nombre de chemins de longueur $N = 3$ est 7.

1) En continuant l'arborescence, combien existe-t-il de chemins de longueur $N = 4$?

2) Faire le programme récursif correspondant à cette arborescence. On ne demande pas d'afficher chaque chemin, même si ce programme s'y prête facilement. On veut seulement connaître le nombre de chemins de longueur *N*, *N* étant donné. Ce nombre sera placé dans une variable *compteur*. Le programme devra utiliser une fonction récursive *arbre*(*i*, *n*) où *i* est le numéro du pas où l'on est, et *n* la longueur du chemin en cours de construction jusqu'à ce pas *i*. Faire le programme principal et la fonction *arbre*().

3) On appelle $u(n)$ la longueur des chemins de longueur *n*. On admettra que ce nombre obéit à la relation de récurrence $u(n+2) = 2\,u(n+1) + u(n)$, avec au départ $u(0) = 1$ et $u(1) = 1$. On veut connaître $u(N)$, *N* étant un nombre donné. Faire le programme qui permet d'avoir $u(N)$. Ce programme devra être itératif, et utiliser seulement trois cases variables, et pas de tableau.

Corrigé :

1) Avec 0 qui admet deux successeurs, 1 qui en admet 3 et 2 qui en admet 2, on trouve 17 chemins pour $N = 4$.

2) Le programme principal se réduit à appeler *arbre*(1, 1), puis à afficher le contenu de *compteur* (variable qui a été déclarée en global, donc automatiquement mise à 0 au départ). La fonction *arbre*() est ainsi construite, sous sa forme la plus rustique :

```
void arbre (int i, int n)
{ if (n ==N) compteur++ ;
  else
    { if (i == 0) { arbre(0, n+1) ; arbre (1, n+1) ;}
      else if (i == 1) { arbre(0, n+1) ; arbre (1, n+1) ; arbre (2, n+1) ;}
      else if (i == 2) { arbre (1, n+1) ; arbre (2, n+1) ;}
    }
}
```

Ou bien cette version complète, et plus concentrée :

```
int main()
{ arbre(1,1);
```

```
printf("N = %d   nombre de formes = %d", N,compteur);
getchar(); return 0;
}

void arbre(int i, int n)
{ int j;
  if (n == N) compteur++;
  else
  { for(j=0;j<3;j++) if (j!= (i+2)%4) arbre(j,n+1);
  }
}
```

Remarque : Si l'on veut dessiner les cheminements sur l'écran, on s'y prend un peu différemment, en faisant intervenir les coordonnées de l'extrémité *x*, *y* de chaque segment qui est construit :

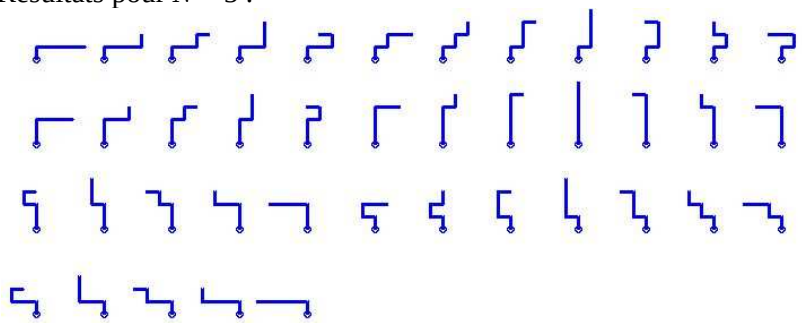
* *Programme principal en C-SDL (sans les déclarations initiales, notamment les variables globales comme x[N+1], etc.) :*

```
xorig=60; yorig=60;
explorer(0,1,1);  /* extrémité de coordonnées 0, 1, pour n = 1*/
SDL_Flip(ecran);pause();  return 0;
```

* *Fonction d'exploration :*

```
void  explorer (int xx, int yy, int n)
{ int i,dx,dy,j,dejapasse,newx,newy;
  if (n<N)
    { for(i=0;i<3;i++)
        { dx=((i+1)%2)*(1-i); dy=(i%2)*(2-i);   newx=xx+dx;newy=yy+dy;
          if (!(newx==x[n-1] && newy==y[n-1])) /* pour éviter un aller-retour */
           { x[n+1]=newx;y[n+1]=newy;
             explorer(newx,newy,n+1);
           }
        }
    }
  else if (n==N)
      {
        x[0]=0;y[0]=0; x[1]=0;y[1]=1;
        circle(xorig+zoom*x[0],yorig-zoom*y[0],3,bleu);
        for(j=0;j<N;j++)
        linewithwidth(xorig+zoom*x[j],yorig-zoom*y[j],xorig+zoom*x[j+1],yorig-zoom*y[j+1],1,bleu);
        compteur++; xorig+=60; if (xorig>750) {yorig+=75; xorig=60; }
        if(yorig>540) { SDL_Flip(ecran);pause(); SDL_FillRect(ecran,0,blanc);
                        xorig=60; yorig=60;
                      }
      }
}
```

Résultats pour *N* = 5 :



3)

```
u = 1 ;
v = 1 ;
for(i = 2 ; i <= N ; i++)
  { w = 2 * v + u ;
    u = v ;
    v = w ;
  }
```

afficher w (ou v)

8) Programmation : des sommets aux arêtes d'un graphe

Un graphe non orienté à *NS* sommets (numérotés de 0 à *NS* – 1) est supposé enregistré sur ordinateur par les listes d'adjacence de ses sommets. Autrement dit, pour chaque sommet *i* on connaît ses voisins *v*[*i*][*j*] ainsi que son nombre de voisins *nbv*[*i*] (*j* allant donc de 0 à *nbv*[*i*] – 1). A partir de là, faire le programme qui donne les arêtes du graphe ainsi que leur nombre *NA*. L'arête numéro *k* sera enregistrée par ses deux extrémités *e1*[*k*] et *e2*[*k*] et l'on choisira *e1*[*k*] < *e2*[*k*].

Corrigé :

```
k = 0 ;
for(i = 0 ; i < NS ; i++)  for(j = 0 ; j < nbv[i] ; j++)
if (i < v[i][j])  { e1[k] = i ; e2[k] = v[i][j] ; k++ ; }
NA = k ;
```