Ex 2 :

* 1. Pour aller de la case (0,0) à la case (2,3) on fait 3 déplacements vers la droite et 2 vers le bas.
  2. Comme on fait des déplacements de 1 pas à chaque étape, il faut faire 2+3=5 déplacements. Chaque déplacement nous amène une nouvelle case. En n’oubliant pas d’inclure la case (0,0) il faut donc parcourir 2+3+1=6 cases.

1. On liste tous les chemins et les sommes associées :

|  |  |
| --- | --- |
| Chemins | Somme |
| (0,0) 🡪 (0,1) 🡪 (0,2) 🡪 (0,3) 🡪 (1,3) 🡪 (2,3) | 11 |
| (0,0) 🡪 (0,1) 🡪 (0,2) 🡪 (1,2) 🡪 (1,3) 🡪 (2,3) | 10 |
| (0,0) 🡪 (0,1) 🡪 (0,2) 🡪 (1,2) 🡪 (2,2) 🡪 (2,3) | 14 |
| (0,0) 🡪 (0,1) 🡪 (1,1) 🡪 (1,2) 🡪 (1,3) 🡪 (2,3) | 9 |
| (0,0) 🡪 (0,1) 🡪 (1,1) 🡪 (1,2) 🡪 (2,2) 🡪 (2,3) | 13 |
| (0,0) 🡪 (0,1) 🡪 (1,1) 🡪 (2,1) 🡪 (2,2) 🡪 (2,3) | 12 |
| (0,0) 🡪 (1,0) 🡪 (1,1) 🡪 (1,2) 🡪 (1,3) 🡪 (2,3) | 10 |
| (0,0) 🡪 (1,0) 🡪 (1,1) 🡪 (1,2) 🡪 (2,2) 🡪 (2,3) | 14 |
| (0,0) 🡪 (1,0) 🡪 (1,1) 🡪 (2,1) 🡪 (2,2) 🡪 (2,3) | 13 |
| (0,0) 🡪 (1,0) 🡪 (2,0) 🡪 (2,1) 🡪 (2,2) 🡪 (2,3) | 16 |

La somme maximale est donc 16.

* 1. Le Tableau T’ est le suivant :

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 5 | 6 | 9 |
| 6 | 6 | 8 | 10 |
| 9 | 10 | 15 | 16 |

* 1. La valeur T’[0][j] où j est non nul correspond à la somme des cases (0,0) à (0,j), c’est-à-dire des cases de la première ligne du tableau.

Il n’y a qu’un seul chemin qui corresponde à cette somme et il passe obligatoirement par la case à gauche (d’indice j-i) de la case (0,j).

Donc pour calculer la somme T’[0][j] on ajoute simplement la valeur de la case (0,j) (c’est-à-dire T[0][j]) à la somme obtenue à la case précédente (c’est-à-dire T’ [0][j-1]).

On a donc bien T’[0][j] = T[0][j]+T’[0][j-1].

1. Si i et j sont non-nuls, il y a deux chemins amenant à la case (i, j). Le premier provient de la case du dessus (i-1,j), le second de la case de gauche (i,j-1).

La valeur de T’[i][j] s’obtient donc en ajoutant la valeur de T[i][j] au maximum des deux chemins menant à cette case : max(T’[i-1][j],T’[i][j-1]).

1. .
   1. Le cas de base est atteint lorsque l’on atteint une case de la première ligne (i vaut 0) ou de la première colonne (j vaut 0). Dans ce cas on calcule la somme en additionnant la valeur de la case en question avec le résultat de somme\_max avec comme argument T est la case précédente (sur la ligne si i=0 ou la colonne si j=0).
   2. On a :

|  |
| --- |
| def somme\_max(T,i,j):  if i==0 and j==0 :  return T[0][0]  elif i==0 :  return T[0][j]+somme\_max(T,0,j-1)  elif j==0 :  return T[i][0]+somme\_max(T,i-1,0)  else :  return T[i][j]+max(somme\_max(T,i-1,j), somme\_max(T,i,j-1)) |

* 1. On appelle somme\_max(T,2,3)

Exercice 3 :

1. La taille d’un arbre est le nombre de nœuds. Ici elle vaut 9. Ka hauteur est la longueur du chemin le plus long entre la racine et l’une des feuilles. Ici 4.
   1. Le numéro associé à G est 1010
   2. Le nombre 13 s’écrit 1101 en binaire. Il correspond donc au nœud I.
   3. Les nœuds les plus bas sont numérotés sur bit (4 dans l’exemple).
   4. Un arbre de hauteur h peut avoir au minimum n nœuds (nœud par niveau). Donc h≤n.

A l’autre extrême, si l’arbre est complet (tous les niveaux sont remplis) alors la racine a pour code 1, les nœuds du premier niveau ont pour code 10 et 11, ceux du deuxième niveau 100, 101, 110 et 111, etc. …

Les codes des nœuds du niveau h s’écrivent sur h bits.

L’arbre étant complet, tous les nombres entier pouvant s’écrire sur h bits correspondent à des codes sauf la valeur 0 car la racine est le 1.

Il y a 2h valeurs possibles sur h bits. Donc en retirant le 0 on obtient 2h-1.

On a donc bien : h≤n≤2h-1.

1. .
   1. Le tableau est :

[None,"A","B","C","D","E","F","G","H","I","J","K","L","M","N","O"]

* 1. L’indice du père d’un nœud d’indice i≥2 est le quotient entier de i par 2(i//2 en python).

1. On propose le code ci-dessous :

|  |
| --- |
| def recherche(arbre, element) :  taille=len(arbre)  i=1  while i<taille :  if abre[i]==element :  return True  elif element<arbre[i] :  i=2\*i  else :  i=2\*i+1  return False |

Exercice 4 :

1. L’attribut num\_eleve est la clé primaire. Elle permet d’identifier de façon certaine des objets de la table;relation.
2. On a :

|  |
| --- |
| INSERT INTO seconde(num\_eleve, langue1,langue2,classe)  VALUES ("133310FE","anglais","espagnol","2A") |

1. On fait :

|  |
| --- |
| UPDATE seconde  SET langue1="allemand"  WHERE num\_eleve = "156929JJ" |

1. .
2. Cette requête renvoie les numéros d’identification de tous les élèves de seconde. Il s’agit donc des données de la première "colonne" du fichier csv
3. Cette requête permet de compter le nombre d’élèves de seconde. Il faudrait disposer de l’ensemble du fichier csv pour pouvoir proposer un résultat plus précis.
4. On fait

|  |
| --- |
| SELECT COUNT(\*) FROM seconde  WHERE langue1 = "allemand" OR langue2="allemand" |

1. L’ajout d’une clé étrangère permet de s’assurer que les données des tables se correspondent. Elle peut aussi permettre d’empêcher d’ajouter des objets dans une table s’ils ne sont pas présents dans l’autre.
2. On fait :

|  |
| --- |
| SELECT nom, prenom, datedenaissance FROM eleve  JOIN seconde ON seconde.num\_eleve=eleve.num\_eleve  WHERE seconde.classe="2A" |

1. On peut faire :

|  |  |  |
| --- | --- | --- |
| coordonnées | | |
| num\_eleve | clé primaire | clé étrangère de la table seconde |
| adresse |  |  |
| code postal |  |  |
| ville |  |  |
| mail |  |  |

Exercice 5 :

1. En lisant la table de routage de A puis celle de C et F on obtient A🡪C🡪F🡪G.
2. On peut avoir :

|  |  |  |
| --- | --- | --- |
| table de routage du routeur G |  |  |
| Destination | Routeur suivant | Distance |
| A | E | 3 |
| B | E | 3 |
| C | E | 2 |
| D | E | 2 |
| E | E | 1 |
| F | F | 1 |

1. On peut avoir :

|  |  |  |
| --- | --- | --- |
| table de routage du routeur A |  |  |
| Destination | Routeur suivant | Distance |
| B | B | 1 |
| D | D | 1 |
| E | D | 2 |
| F | D | 4 |
| G | D | 3 |

1. La liaison ente A et B a un débit de 10Gb/s=1010b/s. Donc sont coût vaut ==0,01.
2. Si le coût est de 5 alors on a = 5 ce qui donne d==20Mb/s.
3. On fournit le graphe du réseau dans la figure1.