Hugo PUC
Charly BEC

M1 Informatique
13/03/2022

Rendu TP2 Knapsack

Question 1

```
int n_rand;
int b_rand;
int* a_rand;
int* c_rand;

n_rand = rand()%101;
b_rand = rand()%1001;

while(b_rand < (int)(7/2*n_rand)) {
        n_rand = rand()%101;
        b_rand = rand()%1001;
}


a_rand = randWeight(n_rand,b_rand);
c_rand = randvalue(n_rand);

write_inst("instance1.csv", a_rand, c_rand, n_rand, b_rand);
```

On évite les poids de 0, avec au moins 3.5 plus de poids que de valeur.

```
int * randWeight(int n, int b)
{
    int temp;
    int moy = (int)(b/n);
    srand(time(NULL));


    int* a_rand = (int*)malloc(sizeof(int)*(n));

    for(int i=0; i<n; i++) {
            temp = rand()%(2*moy)+(moy-((int)moy/2));
            a_rand[i] = temp;
    }

    return a_rand;
}
```

On donne des poids cohérents avec notre valeur max du sac.

```c
int * randvalue(int n)
{
    int * c_rand = (int*)malloc(sizeof(int)*(n));
    srand(time(NULL));

    for(int i=0; i<n; i++) {
        c_rand[i] = rand()%10+1;
    }

    return c_rand;
}
```

Pour les valeurs, un entier entre 1 et 10 suffit.

Question 2

```c
// tri des variables

// tableau tempon c/a, indice i
float** sortArray = (float**)malloc(sizeof(float)*(n*2));

for (int i=0; i<n; i++) {
    sortArray[i] = (float*)malloc(2 * sizeof(float));
}

for (int i=0; i<n; i++) {
    int cs = c[i];
    int as = a[i];
    float result = (float) cs / as;

    sortArray[i][0] = result;
    sortArray[i][1] = i;
}
```

On va tout d'abord créer un tableau double qui va prendre notre ratio poids / valeur en 1ere case et son index dans le tableau en 2eme case.

```
0.142857, 0.000000,
0.555556, 1.000000,
0.111111, 2.000000,
1.666667, 3.000000,
0.200000, 4.000000,
0.263158, 5.000000,
0.818182, 6.000000,
1.000000, 7.000000,
0.142857, 8.000000,
0.400000, 9.000000,
0.333333, 10.000000,
0.200000, 11.000000,
0.263158, 12.000000,
0.181818, 13.000000,
0.166667, 14.000000,
0.200000, 15.000000,
0.666667, 16.000000,
0.047619, 17.000000,
0.157895, 18.000000,
0.200000, 19.000000,
0.400000, 20.000000,
0.714286, 21.000000,
```

Ce qui nous donne un tableau non trié avec les positions.

```c
// fonction pour qsort (a-b: croissant, b-a: décroissant)
int compare (const void * pa, const void * pb)
{
    const int *a = *(const int **)pa;
    const int *b = *(const int **)pb;

    return b[0] - a[0];
}
```

On prépare la fonction compare pour la fonction qsort, on compare nos premières cases sur le tableau double.

```c
// tri le tableau dans l'ordre décroissant c1/a1 ≥c2/a2 ≥…≥ cn/an.
qsort (sortArray, n, sizeof(sortArray[0]), compare);
```

```
1.800000, 39.000000,
1.800000, 65.000000,
1.666667, 3.000000,
1.666667, 58.000000,
1.666667, 71.000000,
1.666667, 92.000000,
1.666667, 93.000000,
1.285714, 83.000000,
1.000000, 7.000000,
1.000000, 25.000000,
1.000000, 72.000000,
0.818182, 6.000000,
0.818182, 82.000000,
0.777778, 40.000000,
0.777778, 73.000000,
0.750000, 85.000000,
0.714286, 21.000000,
```

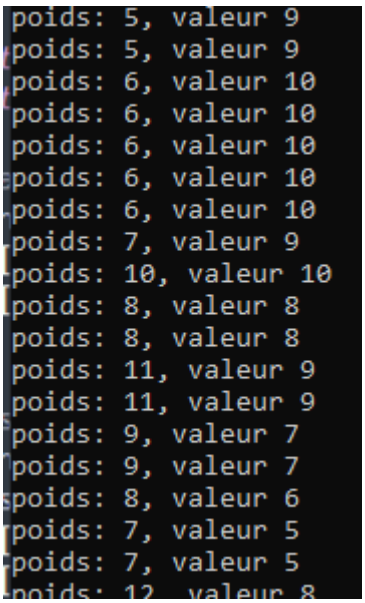
On se retrouve avec un tableau trié par ratio poids / valeurs et de ce couple de valeur transformé en ratio.

```
// tempons tab a et c
int* buffA = (int*)malloc(sizeof(int)*n);
int* buffC = (int*)malloc(sizeof(int)*n);

// recopie des tableaux
for (int i=0; i<n; i++) {
    buffA[i] = a[i];
    buffC[i] = c[i];
}

// ordre décroissant pour le vrai tableau
for (int i=0; i<n; i++) {
    int index = sortArray[i][1];
    a[i] = buffA[index];
    c[i] = buffC[index];
}
```

Nous avons juste à prendre dans l'ordre le tableau trié par ratio et remettre dans l'ordre les tableaux initiaux des couples poids / valeurs.



On retrouve donc les variables triés telles que : c1/a1 ≥c2/a2 ≥…≥ cn/an.

On peut appliquer notre algorithme et afficher la solution x, avec le tableau trié, on se rend compte que les premières variables seront prises.

Question 3

Notre algorithme a une complexité en O(n*b).

Résultat avec une méthode de résolution par programmation dynamique

Capacité du sac	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Boîtes	Valeur du sac															
Valeur	Poids															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	2	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
2	5	0	0	1	1	1	2	2	3	3	3	3	3	3	3	3
3	7	0	0	1	1	1	2	2	3	3	4	4	4	5	5	6	6
7	12	0	0	1	1	1	2	2	3	3	4	4	4	7	7	8	8
10	9	0	0	1	1	1	2	2	3	3	10	10	11	11	11	12	12

Si on fixe n, on va rajouter des colonnes sur cette trace de l'algorithme ci-dessus car b est toujours variable.

Si on fixe b, on va rajouter des lignes à la trace ci-dessus.