IFT2125 Notes

Yuchen Hui 20150470

April 1, 2022

Contents

1 Greedy Algorithm

1.1 Sac à dos greedy version

Proof. Supposons que les objets sont numerotes par ordre decroissant de valeur par unite de poids, i.e.

$$\frac{v1}{w1} \geq \frac{v2}{w2}...$$

par l'algorithme vorace. SI tous les $x_i = 1$,alors la solution est trivialement optimale.

Sinon,soit j le plus petit indice tel que $x_j < 1$, on a alors que $x_i = 1, \forall i < j$ et $x_i = 0, \forall i > j$ et $\sum_{i=1}^{n} x_i w_i = W$.

Soit $V(x) \sum_{i=1}^{n} x_i v_i$, la valeur de la soluttion X on doit demontrer que $V(x)$ est maximale.

Soit $Y = (x_1, x_2, ..., x_n)$ une autre solution de probleme et soit $V(y)$ sa valeur. comme Y est une solution , $0 \leq y_i \leq 1, \forall i et \sum_{i=1}^{n} y_i w_i = W$

On veut montrer que $V(x) - V(y) \geq 0$ et alors X sera la solution optimale Soit j le plus petit indice tel que $x_j < 1$ si i <j, alors $\frac{v_i}{w_i} \geq \frac{v_j}{w_j} et x_i = 1$

□

1.2 File d'attente

Strategie vorace: classe les clients par ordre croissant des ti et executer les taches dans cet ordre

Proof. Soit n clients ordonne arbitrairement est servit selon l'ordre c = 1,2,3,..,n

Le temps total de service requit est $T(c) = t_1 + (r + 1 + t_2) \dots + = nt_1 + (n-1)t_2 + \dots + t_n$

preuve par contradiction: i.e. qu<on suppose $T(c)$ est optimal et c n'est pas l'ordre dans lequel on sert les clients en ordre croissant des t_i $\square$

1.3 Kruscal Algo

1.3.1 Matrix Version Algo

nothing here

2 Programmation dynamique

2.1 Knapsack Problem 2

Time complexity: $\mathcal{O}(nW)$ for constructing the matrix(table) and $\mathcal{O}(n+w)$for decomposition of the optimal load.

Et voici les codes:

Algorithm 1 fonction knapsack_dy($w[1..n], v[1..n], W$): **array** V[0..n, 0..W]

Require: $v_i > 0, w_i > 0, x_i \in \{0,1\}, W \in \mathbb{N}^*$

{array $w[1..n]$ indicates weights of objects 1 to n, array $n[1..n]$ indicates their values. W is the max weight a knapsack can bear. Here come initialisations:}

array $w[1..n]$ =???

array $v[1..n]$ = ???

array $V[0..n, 0..W]$

for $j = 1$ **to** W **do** $V[0,j] = 0$

{establish matrix}

for $i = 1$ **to** n **do**

  for $j = 1$ **to** W **do**

    $V[i,j] \leftarrow$ **if** $j - w[i] < 0$ **then** $V[i-1,j]$

      else $V[i,j] = \max(V[i-1,j], V[i-1, j-w[i]] + v[i])$

  end for

end for

return $V[0..n, 0..W]$

```python

import time
import sys

# function from Marc feely
```

```python
def createMatrix(numRow,numColumn):
    result = [None] * numRow
    for i in range(numRow):
        result[i] = [0]*numColumn
    return result


# 1.     weightvalueindex -1,     pythonlistindex0
# 2. matrix include line 0 ( a line filled with 0), but the matrix
#    in manual don't have line 0 but begin with line 1
def knapsack_dynamic(weights, values, W_max, waitTime):
    num_objects = len(weights)
    V = createMatrix(len(weights)+1,W_max+1)
    for j in range(1,W_max+1): V[0][j] = 0
    for i in range(1,num_objects+1):
        for j in range(0, W_max+1):
            if (j-weights[i-1] < 0):
                V[i][j] = V[i-1][j]
            else:
                V[i][j] = max(V[i-1][j],
                              V[i-1][j-weights[i-1]]+values[i-1])

            print(V[i][j], "\t", end = "")
            sys.stdout.flush()
            time.sleep(waitTime)
        print("\n")


    print(V)

#If we have infinite number of objects,
def knapsack_dynamic_infinity(weights, values, W_max, waitTime):
    num_objects = len(weights)
    V = createMatrix(len(weights)+1,W_max+1)
    for j in range(1,W_max+1): V[0][j] = 0
    for i in range(1,num_objects+1):
        for j in range(0, W_max+1):
            if (j-weights[i-1] < 0):
                V[i][j] = V[i-1][j]
            else:
                V[i][j] = max(V[i-1][j],
                              V[i-1][j-weights[i-1]]+values[i-1],
                              V[i][j-weights[i-1]]+values[i-1])

            print(V[i][j], "\t", end = "")
            sys.stdout.flush()
            time.sleep(waitTime)
        print("\n")


    print(V)
knapsack_dynamic([1,2,5,6,7],[1,6,18,22,28],11,0)
knapsack_dynamic_infinity([1,2,5,6,7],[1,6,18,22,28],11,0)
```

Listing 1: codes pour knapsack2

2.2 Shortest paths (Floyd)

Time complexity : $\mathcal{O}\left(n^3\right)$

Algorithm 2 fonction Floyd($L[1..n, 1..n]$)**: array** $D[0..n, 0..n]$

```
array D[1..n, 1..n]
D ← L
for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            D[i, j] ← min (D[i, j], D[i, k] + D[k, j])
return D
```

3 Divide and Conquer

DEMO 4.1