IFT2125 Notes

Yuchen Hui 20150470

March 15, 2022

Contents

1 Kruscal Algo **1**
1.1 Matrix Version Algo 1

2 Greedy Algorithm **1**
2.1 Sac à dos greedy version 1
2.2 File d'attente 2

3 Programmation dynamique **2**
3.1 Knapsack Problem 2 2

1 Kruscal Algo

1.1 Matrix Version Algo

2 Greedy Algorithm

2.1 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 , $0 \leq y_i \leq 1, \forall i \text{et} \sum_{i=1}^{n} y_i w_i = W$

Algorithm 1 Kruskal

Require: $n \geq 0 \vee x \neq 0$
Ensure: $y = x^n$
  $y \leftarrow 1$
  if $n < 0$ **then**
    $X \leftarrow 1/x$
    $N \leftarrow -n$
  else
    $X \leftarrow x$
    $N \leftarrow n$
  end if
  while $N \neq 0$ **do**
    if N is even **then**
      $X \leftarrow X \times X$
      $N \leftarrow N/2$
    else $\{N$ is odd$\}$
      $y \leftarrow y \times X$
      $N \leftarrow N - 1$
    end if
  end while

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 $\frac{v_i}{w_i} \geq \frac{v_j}{w_j} et x_i = 1$

□

2.2 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 $T(c) = t_1 + (r + 1 + t_2) ... + = nt_1 + (n-1)t_2 + ... + t_n$

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 □

3 Programmation dynamique

3.1 Knapsack Problem 2

Algorithm 2 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 sac a dos can bear. Here comes initialisation}

```
array w[1..n] = ???
array v[1..n] = ???
array V[0..n,0..W]
for j = 1 to W do V[0, j] = 0
for i = 1 to n do
  for j = 1 to W do
    V[i, 0] ← if j < 0 then -1
              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]
```