KNAPSACK

1

1) a) Putem asocia această problemă cu problema Knapsack 0/1 astfel:

K = capacitatea rucsacului

$s_i, \forall i = \overline{1,n} = \begin{cases} i = \text{obiectul} \\ s_i = \text{greutatea lui}, s_i = \text{valoarea lui} \end{cases}$

Vom folosi metoda programării dinamice:

$d[i][w]$ = valoarea maximă din primele i obiecte cu greutatea totală $\leq w$

$$d[i][w] = \max \begin{cases} d[i-1][w] \\ d[i-1][w - s_i] + s_i \end{cases}$$

La final răspunsul va fi: $d[n][K]$

Complexitate timp: $O(n \cdot K)$ -> calculăm matricea

Complexitate spațiu: $O(n \cdot K)$ -> reținem matricea

Algoritmul oferă soluția optimă deoarece la bază se află rezolvarea recursivă, generând toate metodele de a alege cele n obiecte => 2^n. Dar, pentru că multe din subprobleme rezolvării recursive se repetă, putem să memoizăm rezultatele în matricea d.

b) Idee:

Adunăm într-o variabilă temp numere cât timp temp ≤ K.

Când temp > K, vedem care e mai mare decât K/2: temp sau elementul, și îl luăm pe acela

```
temp = 0
for nr in S:
    if temp + nr <= k:
        temp = temp + nr
    else if temp < x:
        temp = x
```

aici știm că suma lor este mai mare decât K deci vrem elementul maxim

Scanat cu CamScanner

LOAD BALANCE

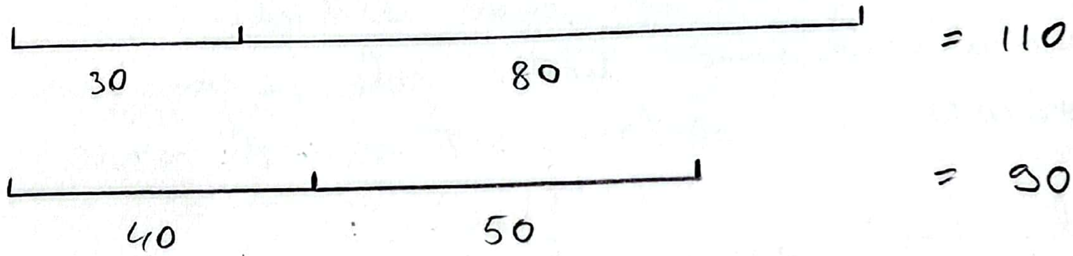
0,5

1) a)

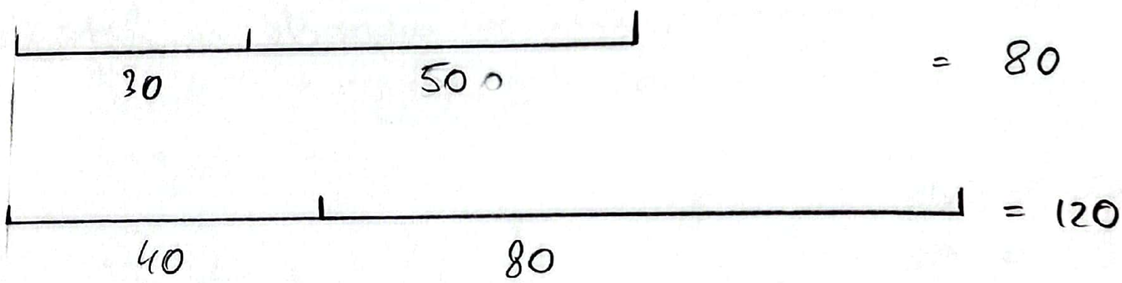
Propunem setul următor de 4 activități:

30, 40, 50, 80

OPT:



ALG:



$ALG \le 1.1 \cdot OPT$

$120 \le 121$ => algoritmul studentului poate fi 1.1 - aproximativ

0,5

b) Să presupunem că setul de activități se poate distribui egal către cele 2 mașini, astfel având OPT = 100

ALG / 100 = 120 / 100 = 1.2 => 1.1 =>

=> nu este 1.1 - aproximativ

Dar, dacă nu ar fi egal distribuite, diferența maximă dintre cele 2 mașini la oricare pas pentru OPT va fi <= 10

Astfel, dacă avem S = 200 și a, b a.î.

$|a - b| <= 10$

$a + b = 200$

a = maxim, a.î. să respecte condițiile de mai sus

=>

=> $a = 105$

$b = 95$

Deci, 120 / 105 = 1,14 > 1,1 =>

=> tot nu este 1.1.-aproximativ

3) Preluăm notațiile de la curs:
k = indicele mașinii cu load-ul maxim
q = ultimul job al mașinii k
t_q = timpul job-ului q
$load'(i)$ = load-ul mașinii i după ce am asignat $q-1$ job-uri

2

$$ALG = load'(k) + t_q$$

$$load'(k) \le \frac{1}{m} \sum_{i=1}^{m} load'(i) =$$

$$= \frac{1}{m} \sum_{j=1}^{q} t_j + t_q \le \frac{1}{m} \left(\sum_{j=1}^{m} t_j - t_q \right) + t_q = \frac{1}{m} \sum_{j=1}^{m} t_j - \frac{1}{m} +$$

$$= \frac{1}{m} \sum_{j=1}^{m} t_j + t_q \left(1 - \frac{1}{m}\right) =$$

$$= \frac{1}{m} \sum_{j=1}^{m} t_j + \frac{1}{2} (t_m + t_{m+1}) \left(1 - \frac{1}{m}\right) =$$

$$= \frac{1}{m} \sum_{j=1}^{m} t_j + \left(\frac{1}{2} - \frac{1}{2m}\right)(t_m + t_{m+1}) \Leftrightarrow$$

$$OPT \ge t_m + t_{m+1}$$

$$\Leftrightarrow OPT + \left(\frac{1}{2} - \frac{1}{2m}\right) OPT = OPT \left(1 + \frac{1}{2} - \frac{1}{2m}\right) =$$

$$= \frac{3}{2} - \frac{1}{2m} OPT$$

Scanat cu CamScanner

TRAVELLING SALESMAN

1

1) a) Presupunem prin absurd că problema noastră nu se află în NP Hard.

Plecăm de la graful $G(V, E)$ și construim graful $G'(V, E')$ astfel:

- pentru fiecare $e \in E'$ pentru care $e \in E$ avem $W(e, G') = 1$
- pentru fiecare $e \in E'$ pentru care $e \in E$ avem $W(e, G') = 2$

Sau pe scurt, muchiile care există în graful G au costul 1 și cele care nu, costul 2.

Graful G' va fi un graf complet.

- Dacă G are ciclu hamiltonian =>
  => există ciclu hamiltonian și în G' de cost n, unde $n = |V|$ (pentru că toate muchiile vor avea ponderea 1) ①
- Dacă G nu are ciclu hamiltonian =>
  => soluția va avea cost minim $(n-1)+2 = n+1$ (va folosi cel puțin o muchie din G') ②

Astfel, Graful G' se obține în timp polinomial, algoritmul rulează în timp polinomial.

①, ② -> decizarea ciclului hamiltonian este tot polinomială (decidem din output-ul algoritmului nostru)

dar decizarea ciclului hamiltonian e NPC =>

=> Din ipoteza propusă și contradicția la care am ajuns => Concluzia: Problema rămâne în NPHard pentru această instanță

b) Având doar muchii cu ponderi 1, 2 le putem testa pe toate:

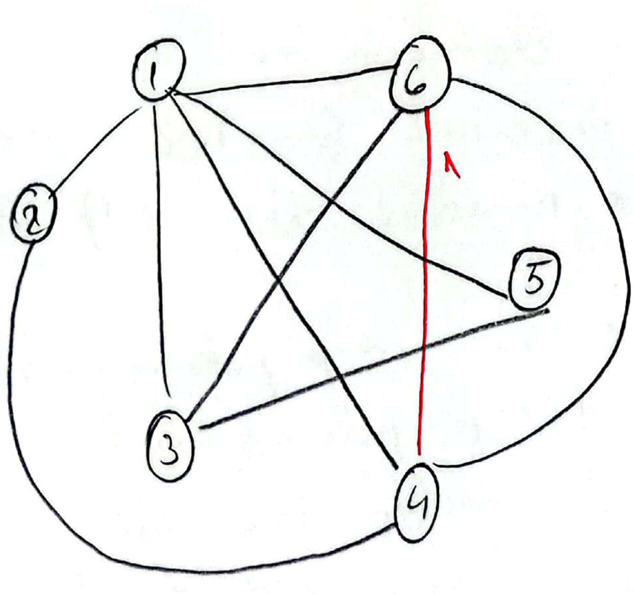
$$\left.\begin{array}{lll} 1, 1, 1 & \Rightarrow & 1+1 \geq 1 \\ 1, 1, 2 & \Rightarrow & 1+1 \geq 2 \\ 1, 2, 2 & \Rightarrow & 1+2 \geq 2 \\ 2, 2, 2 & \Rightarrow & 2+2 \geq 2 \end{array}\right| \Rightarrow$$

=> Ponderile satisfac inegalitatea triunghiului

c) În primul rând, putem aplica algoritmul din curs pentru că ponderile respectă inegalitatea triunghiului (b)

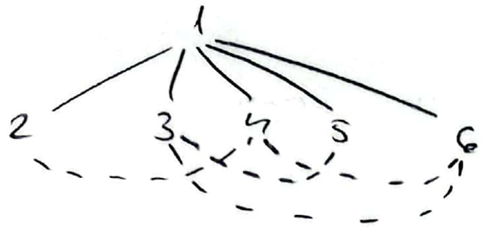
Presupunem că pentru această instanță algoritmul este $\frac{3}{2}$-aproximativ.

Fie graful:



- toate muchiile au ponderea 1
- restul muchiilor care nu se află, dar sunt necesare pentru a fi graf complet vor avea ponderea 2

Arborele MST :



- muchiile solide aparțin MST
- muchiile punctate nu aparțin MST

1

OPT : 1, 2, 4, 6, 3, 5, 1 = 6

MST : 1, 2, 1, 3, 1, 4, 1, 5, 1, 6, 1 = 10 ⇔

⇔ 1, 2, 3, 4, 5, 6, 1

10 / 6 = 1,66 > 1,5 ⇒

⇒ În această instanță algoritmul din curs nu este $\frac{3}{2}$ - aproximativ.

VERTEX COVER

a) Cazul worst case: 2

$$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_4 \vee x_5) \wedge \dots \wedge (x_1 \vee x_{m-1} \vee x_m)$$

OPT = alegem x_1

ALG = ar putea alege mereu unul din cei 2 termeni rămași astfel alegând m termeni

Factor = m-aproximativ unde m e nr. de clauze

b) În loc să alegem o singură variabilă din C_j, le luăm pe toate 3 și le setăm la TRUE.

iar în cel mai rău caz, alg. nostru va alege de trei ori mai multe variabile decât OPT.

c) V = vector binar = $[x_1, \dots, x_m]$

$$x_i = \begin{cases} 1 \Rightarrow x_i = true \\ 0 \Rightarrow x_i = false \end{cases}$$

minimizăm funcția $\sum_{i=1} x_i$

Constrângeri

$x_i + x_j + x_k \geq 1$, $\forall C_t$, $t = \overline{1,m}$

$0 \leq x_i \leq 1$, $\forall i \in \{1, m\}$

$\Rightarrow$ cu threshol $\frac{1}{3}$

d) setăm threshold-ul $\frac{1}{3}$ astfel

$$\begin{cases} x_i \leq \frac{1}{3} \Rightarrow x_i = 1 \\ x_i \geq \frac{1}{3} \Rightarrow x_i = 0 \end{cases}$$

Dacă avem ghinion putem alege toți $x_i, i \in \overline{1,n}$

și ne întoarcem la a) $\Rightarrow$

$\Rightarrow$ 3-aproximativ