Lazăr Mihai, grupa 234

Knapsack

a)
```
int maxSum(int k, vector<int> &s){
   vector<int> v(k+1, 0);
   for(auto e : s){
      for(int g = k; g >= e; --g){
         v[g] = max(v[g], v[g-e] + e);
      }
   }
   return v[k];
}
```

b)
```
int maxSum2(ifstream &fin){
   int k, e, res = 0;
   fin>>k;
   while(fin>>e){
      if(e + res <= k){
         res += e;
      }
      else if(res < e){
         res = e;
      }
   }
   return res;
}
```

Pentru a ajunge la o solutie $\frac{1}{2}$ OPT, vom aduna toate numerele din șir fără a depăși valoarea k, astfel vom avea două cazuri:

Caz 1: Suma noastră totală ajunge la res $\geq \frac{k}{2}$, condiția fiind îndeplinită.

Caz 2: Suma noastră totală ajunge la res $\leq \frac{k}{2}$, iar urmatorul element $s_i \geq k - \frac{k}{2}$, adică $s_i \geq \frac{k}{2}$, caz în care res devine s_i, condiția fiind îndeplinită.

Lazăr Mihai, grupa 234

Load Balance

1 A:
Fie setul {20, 60, 30, 90}.
Soluția optimă va fi max({60, 30}, {90, 20}) = 110.
Soluția propusă de student va fi max({60, 20}, {90, 30}) = 120.
Știind că 110 * 1.1 = 121, 121 > 120 => algoritmul propus de student poate fi 1.1 aproximativ.

1 B:
În cazul în care activitățile au timpul de lucru ≤ 10, algoritmul optim va aloca activitățile echilibrat, generând o diferență maximă dintre cele 2 încărcături ≤ 10.
Pentru a putea fi un algoritm 1.1 aproximativ, diferența dintre cele 2 îcărcături trebuie să fie $\leq 10 * 1.1 = 11$, iar diferența încărcături propuse de student este 120 – 80 = 40 > 11, deci algoritmul nu poate fi 1.1 aproximativ.

3:
Fie k indicele masinii cu load maxim in urma executarii algoritmului.
Fie q ultimul job adaugat masinii k.
fie load'(M) - load-ul masiniii M dupa ce am asignat primele q-1 joburi dar nu si jobul q

$OPT \geq \max\{\frac{1}{m}\sum_{1\leq j\leq n} t_j, \max\{t_j | 1\leq j \leq n\}\}$
$load'(M_k) \leq \frac{1}{m}\sum_{1\leq i\leq m} load'(M_i) \leq \frac{m+1}{2m}\sum_{1\leq i\leq m} load'(M_i) \leq \frac{m+1}{2m}\sum_{1\leq j<q} t_j < \frac{m+1}{2m}\sum_{1\leq j\leq n} t_j \leq$
$\leq LB \leq OPT$
$ALG = load'(M_k) + t_q \leq LB + t_q \leq \max\{t_j | 1\leq j \leq n\} + LB \leq OPT + OPT \leq 2 \times OPT$
$load'(M_k) \leq \frac{m+1}{2m}\sum_{1\leq i\leq m} load'(M_i) \leq \frac{m+1}{2m}\sum_{1\leq j<q} t_j \leq \frac{m+1}{2m}(\sum_{1\leq j\leq n} t_j - t_q) \leq \frac{m+1}{2m}\sum_{1\leq j\leq n} t_j -$
$- \frac{m+1}{2m} t_q$

$ALG = load'(M_k) + t_q \leq \frac{m+1}{2m}\sum_{1\leq j\leq n} t_j - \frac{m+1}{2m} t_q + t_q \leq OPT - \frac{m+1}{2m} t_q + t_q \leq OPT - \frac{m+1}{2m} t_{max} + t_{max} \leq$
$OPT - \frac{m+1}{2m} OPT + OPT = 2\ OPT - \frac{m+1}{2m} OPT = \frac{4m-m-1}{2m} OPT = \frac{3m-1}{2m} OPT = (\frac{3}{2} - \frac{1}{2m})\ OPT =>$
=> Algoritmul poate fi imbunatatit la $(\frac{3}{2} - \frac{1}{2m})$ OPT

Lazăr Mihai, grupa 234

TSP

1 A:

Presupunem că pentru această situație TSP nu este NP-Hard.

Fie graful G(V, E) și graful G'(V',E') unde muchiile au costul 1 daca acestea aparțin grafului G și costul 2 dacă nu aparțin.

Putem observa că graful G' îndeplinește condiția impusă de noi, adică costul muchiilor este 1 sau 2.

Soluția optimă, prin aplicarea algoritmului TSP, va avea costul total minim N(nr de noduri) doar dacă există ciclu hamiltonian în graful G.

Astfel algoritmul propus de noi pe graful G' se va reduce la algoritmul de aflare al unui ciclu hamiltonian. Evident acest algoritm este NP-Hard, deci presupunera noastră nu este corectă.

Concluzie: Algoritmul TSP pe cazul dat este NP-Hard.

1 B:

Pentru a verifica inegalitate triunghiului avem următoarele cazuri:

{1, 1, 1} $1 \leq 1 + 1$
{1, 1, 2} $2 \leq 1 + 1$
{1, 2, 2} $2 \leq 1 + 2$
{2, 2, 2} $2 \leq 2 + 2$

Concluzie: Acești ponderi satisfac inegalitatea triunghiului.

1 C:

Fie graful complet H(V, E) unde toate muchiile au costul 1.

Fie A(V, E) arborele parțial de cost minim asociat grafului H.

Știm că algoritmul descris în curs(c3, slides 18-19) parcurge muchiile APM-ului de exact două ori, adică 2(n-1) = 2n – 2.

Prin urmare, algoritmul nu poate fi $\frac{3}{2}$ aproximativ, deoarece $2n - 2 > \frac{3}{2} n$, $\forall$ n > 4.

Lazăr Mihai, grupa 234

Vertex Cover

A:

Fie C = (x_1 ⋁ x_2 ⋁ $\mathbf{x_3}$) ⋀ (x_1 ⋁ $\mathbf{x_4}$ ⋁ x_2) ⋀ (x_2 ⋁ x_5 ⋁ $\mathbf{x_6}$)⋀ ($\mathbf{x_7}$ ⋁ x_5 ⋁ x_2)

Pentru algoritmul propus, se observa că în cel mai rău caz, am putea alege un x_i pentru fiecare mulțime care nu apare în nicio altă mulțime (exemplu x_i -urile boldate), deci în acest caz algoritmul este n-aproximativ.

B:

1: C = {C_1, . . . , C_m} mulțimea de predicate, X = {x_1, . . . , x_n} - mulțime de variabile
2: cât timp C ≠ ∅ execută

3: Alegem aleator $C_j \in$ C.

4: $x_i \leftarrow$ true, $\forall x_i \in C_j$.

5: Eliminăm din C toate predicatele ce îl conțin pe x_i, $\forall x_i \in C_j$.

6: return X

Fie mulțimea C = ($x_1 \vee x_2 \vee x_3$) ∧ ($x_4 \vee x_5 \vee x_6$) ∧ **…** ∧ ($x_{n-2} \vee x_{n-1} \vee x_n$), știm că în cel mai rău caz algoritmul va selecta 3 variabile la fiecare pas, adică algoritmul optim ar alege toate valorile x_i , unde $i \in \{3, 6, 9, \ldots, n\}$, adică $\mathrm{OPT} = \frac{n}{3}$, iar $\mathrm{ALG} = \mathrm{n}$, $\Rightarrow \mathrm{ALG} \leq 3\mathrm{OPT} \Rightarrow$ Alogritmul propus este 3-aproximativ.

C:

Fie X = {$x_1, x_2, x_3, \ldots, x_n$} și C = {$C_1, C_2, C_3, \ldots, C_m$}, unde:

1. $0 \leq x_i \leq 1, \forall i \in \{1, 2, \ldots, n\}$
2. $\forall C_i \in C$ și $x_{i1}, x_{i2}, x_{i3} \in C_i$, $x_{i1} + x_{i2} + x_{i3} \geq 1$

Minimizați suma: $\sum_{1 \leq i \leq n} x_i$

D:

$$\mathrm{ALG} = \sum_{1 \leq i \leq n} \begin{cases} 1, & x_i \geq \frac{1}{3} \\ 0, & x_i < \frac{1}{3} \end{cases} \leq \sum_{1 \leq i \leq n} x_i \leq \sum_{1 \leq i \leq n} 3x_i \leq 3 \sum_{1 \leq i \leq n} x_i \leq 3OPT$$