Design and Analysis of Algorithms

Mohammad GANJTABESH

mgtabesh@ut.ac.ir

School of Mathematics, Statistics and Computer Science,
University of Tehran,
Tehran, Iran.

Techniques for the design of Algorithms

The classical techniques are as follows:

1. Divide and Conquer
2. Dynamic Programming
3. Greedy Algorithms
4. Backtracking Algorithms
5. Branch and Bound Algorithms

Greedy Algorithms

- Always make the choice that looks best at the moment.
- They make a locally optimal choice in the hope that this choice will lead to a globally optimal solution.
- They have not any opportunity to go back and fix the partial solution.
- Do not always yield optimal solutions, but for many problems they do.

Greedy Algorithms

Suppose that the following functions are available:

- Feasible
- Optimal

Greedy Algorithms

Suppose that the following functions are available:

- Feasible
- Optimal

Any Greedy Algorithm can be expressed as:

$$
\begin{array}{l}
\mathit{Greedy} - \mathit{Algorithm}\ (\mathit{Set}\ C)\{ \\
\quad S \leftarrow \emptyset; \\
\quad \mathit{While}\ (\mathit{not}\ \mathit{Solution}(S)\ \mathit{and}\ C \neq \emptyset)\{ \\
\quad\quad x \leftarrow \mathit{Select}(C); \\
\quad\quad C \leftarrow C - \{x\}; \\
\quad\quad \mathit{if}\ (\mathit{Feasible}(S \cup \{x\}))\{ \\
\quad\quad\quad S \leftarrow S \cup \{x\}; \\
\quad\quad \} \\
\quad \} \\
\quad \mathit{if}\ (\mathit{Solution}(S))\ \mathit{return}(S); \\
\quad \mathit{else}\ \mathit{return}(''\ \mathit{nosolution}''\ ); \\
\}
\end{array}
$$

Job Scheduling Problem

Definition

- Given a set $J = \{j_1, j_2, \cdots, j_n\}$ of n jobs.
- Each job j_i requires time t_i to be finished.
- The jobs are done one by one.

The Job Scheduling Problem is to arrange the jobs in such a way that the overall waiting time in any system to finish all jobs becomes minimum, i.e.

$$\textit{minimize} \sum_{\ell=1}^{n} (\textit{time in system for job } j_\ell)$$

Do you have any idea...?

Job Scheduling Problem: Example

Example

Suppose that $t_1 = 5$, $t_2 = 10$ and $t_3 = 3$. The following arrangements are possible:

- $\langle t_1, t_2, t_3 \rangle \Longrightarrow T = 5 + (5 + 10) + (5 + 10 + 3) = 38$
- $\langle t_1, t_3, t_2 \rangle \Longrightarrow T = 5 + (5 + 3) + (5 + 3 + 10) = 31$
- $\langle t_2, t_1, t_3 \rangle \Longrightarrow T = 10 + (10 + 5) + (10 + 5 + 3) = 43$
- $\langle t_2, t_3, t_1 \rangle \Longrightarrow T = 10 + (10 + 3) + (10 + 3 + 5) = 41$
- $\langle t_3, t_1, t_2 \rangle \Longrightarrow T = 3 + (3 + 5) + (3 + 5 + 10) = 29\surd$
- $\langle t_3, t_2, t_1 \rangle \Longrightarrow T = 3 + (3 + 10) + (3 + 5 + 10) = 35$

Job Scheduling Problem

The following algorithm solves the Job Scheduling problem:

```
Greedy-Job-Scheduling-Algorithm(I = ⟨t_1, t_2, ···, t_n⟩){
    Sort I in nondecreasing order, say I = ⟨t_{i_1}, t_{i_2}, ···, t_{i_n}⟩;
    // Perform the jobs with respect to the sorted arangement;
    While(I ≠ ∅){
        x ← First element of I;
        I ← I − {x};
        Performs the job x;
    }
}
```

Job Scheduling Problem

The following algorithm solves the Job Scheduling problem:

Greedy-Job-Scheduling-Algorithm$(I = \langle t_1, t_2, \cdots, t_n \rangle)\{$

  Sort I in nondecreasing order, say $I = \langle t_{i_1}, t_{i_2}, \cdots, t_{i_n} \rangle$;

  // Perform the jobs with respect to the sorted arangement;

  While$(I \neq \emptyset)\{$

    $x \leftarrow$ First element of I;

    $I \leftarrow I - \{x\}$;

    Performs the job x;

  $\}$

$\}$

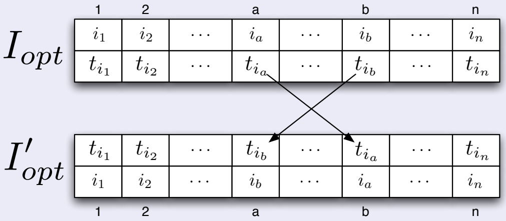
Theorem

The above algorithm solves the job scheduling problem in such a way that the overall waiting time in any system is minimum.

Job Scheduling Problem

Proof.

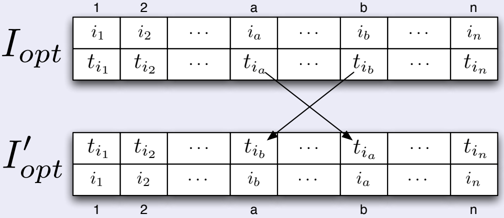
$T(I) = \sum_{k=1}^{n} (n-k+1)t_{i_k}$, if I_{opt} is not sorted, then $\exists a, b : a < b$ and $t_{i_a} > t_{i_b}$



Job Scheduling Problem

Proof.

$T(I) = \sum_{k=1}^{n} (n-k+1)t_{i_k}$, if I_{opt} is not sorted, then $\exists a, b : a < b$ and $t_{i_a} > t_{i_b}$



$$
\begin{aligned}
T(I_{opt}) &= \sum_{k=1}^{n} (n-k+1)t_{i_k} \\
T(I'_{opt}) &= \sum_{\substack{k=1 \\ k \neq a,b}}^{n} (n-k+1)t_{i_k} + (n-a+1)t_{i_b} + (n-b+1)t_{i_a}
\end{aligned}
$$

Job Scheduling Problem

Proof.

$T(I) = \sum_{k=1}^{n} (n-k+1)t_{i_k}$, if I_{opt} is not sorted, then $\exists a, b : a < b$ and $t_{i_a} > t_{i_b}$

I_{opt}

1	2		a		b		n
i_1	i_2	$\cdots$	i_a	$\cdots$	i_b	$\cdots$	i_n
t_{i_1}	t_{i_2}	$\cdots$	t_{i_a}	$\cdots$	t_{i_b}	$\cdots$	t_{i_n}

I'_{opt}

t_{i_1}	t_{i_2}	$\cdots$	t_{i_b}	$\cdots$	t_{i_a}	$\cdots$	t_{i_n}
i_1	i_2	$\cdots$	i_b	$\cdots$	i_a	$\cdots$	i_n
1	2		a		b		n

$$T(I_{opt}) = \sum_{k=1}^{n} (n-k+1)t_{i_k}$$

$$T(I'_{opt}) = \sum_{\substack{k=1 \\ k \neq a,b}}^{n} (n-k+1)t_{i_k} + (n-a+1)t_{i_b} + (n-b+1)t_{i_a}$$

$$\implies T(I_{opt}) - T(I'_{opt}) > 0. \text{ (Contradiction)}$$

$\square$