# Announcements

- Weekly Reading Assignment: Chapter 16
- Homework 5 due today
- Homework 6 due on Thursday, Dec. 1, 2005
- Final Review Session will be held in SN115 on Dec. 12, 2005

# Optimization Problems

- In which a set of choices must be made in order to arrive at an optimal solution, subject to some constraints. (There may be several solutions to achieve the optimal value.)

- Two common techniques:
  - Dynamic Programming (global)
  - Greedy Algorithms (local)

# Intro to Greedy Algorithms

Greedy algorithms are typically used to solve optimization problems & normally consist of

- Set of *candidates*
- Set of candidates that have already been used
- Function that checks whether a particular set of candidates *provides a solution* to the problem
- Function that checks if a set of candidates is *feasible*
- *Selection function* indicating at any time which is the most promising candidate not yet used
- *Objective function* giving the value of a solution; this is the function we are trying to optimize

# Step by Step Approach

- Initially, the set of chosen candidates is empty
- At each step, add to this set the best remaining candidate; this is guided by selection function.
- If enlarged set is no longer feasible, then remove the candidate just added; else it stays.
- Each time the set of chosen candidates is enlarged, check whether the current set now constitutes a solution to the problem.

*When a greedy algorithm works correctly, the first solution found in this way is always optimal.*

# Greedy($C$)

// $C$ is the set of all candidates

$S \leftarrow \varnothing$ // $S$ is the set in which we construct solutions

while not *solution*($S$) and $C \neq \varnothing$ do

  $x \leftarrow$ an element of $C$ maximizing *select*($x$)

  $C \leftarrow C \setminus \{x\}$

  if *feasible*($S \cup \{x\}$) then $S \leftarrow S \cup \{x\}$

if *solution*($S$) then return $S$

  else return "*there are no solutions*"

# Analysis

- The selection function is usually based on the objective function; they may be identical.  But, often there are several plausible ones.
- At every step, the procedure chooses the best morsel it can swallow, without worrying about the future.  It never changes its mind:  once a candidate is included in the solution, it is there for good; once a candidate is excluded, it's never considered again.
- Greedy algorithms do NOT always yield optimal solutions, but for many problems they do.

# Examples of Greedy Algorithms

- Scheduling
  - Activity Selection (Chap 17.1)
  - Minimizing time in system
  - Deadline scheduling
- Graph Algorithms
  - Minimum Spanning Trees (Chap 24)
  - Dijkstra's (shortest path) Algorithm (Chap 25)
- Other Heuristics
  - Coloring a graph
  - Traveling Salesman (Chap 37.2)
  - Set-covering (Chap 37.3)

# Elements of Greedy Strategy

- ***Greedy-choice property*: A global optimal solution can be arrived at by making locally optimal (greedy) choices**
- ***Optimal substructure*: an optimal solution to the problem contains within it optimal solutions to sub-problems**
  - **Be able to demonstrate that if $A$ is an optimal solution containing $s_1$, then the set $A' = A - \{s_1\}$ is an optimal solution to a smaller problem w/o $s_1$. (See proof of Theorem 16.1)**

# Knapsack Problem

- ***0-1 knapsack***: A thief robbing a store finds $n$ items; the $i$th item is worth $v_i$ dollars and weighs $w_i$ pounds, where $v_i$ and $w_i$ are integers. He wants to take as valuable a load as possible, but he can only carry at most $W$ pounds. What items should he take?
- ***Fractional knapsack***: Same set up. But, the thief can take fractions of items, instead of making a binary (0-1) choice for each item.

# Comparisons

- Which problem exhibits greedy choice property?

- Which one exhibits optimal-substructure property?

# Minimizing Time in the System

- A single server (a processor, a gas pump, a cashier in a bank, and so on) has $n$ customers to serve. The service time required by each customer is known in advance: customer $i$ will take time $t_i$, $1 \leq i \leq n$. We want to minimize

$$T = \sum_{i = 1 \text{ to } n} (\text{time in system for customer } i)$$

# Example

- We have 3 customers with

$$t_1 = 5,\quad t_2 = 10,\quad t_3 = 3$$

| *Order* | *T* | | |
|---|---|---|---|
| 1 2 3: | 5 + (5+10) + (5+10+3) | = 38 | |
| 1 3 2: | 5 + (5+3) + (5+3+10) | = 31 | |
| 2 1 3: | 10 + (10+5) + (10+5+3) | = 43 | |
| 2 3 1: | 10 + (10+3) + (10+3+5) | = 41 | |
| 3 1 2: | 3 + (3+5) + (3+5+10) | = 29 | ← optimal |
| 3 2 1: | 3 + (3+10) + (3+10+5) | = 34 | |

# Designing Algorithm

- Imagine an algorithm that builds the optimal schedule step by step. Suppose after serving customer $i_1, \ldots, i_m$ we add customer $j$. The increase in $T$ at this stage is

$$t_{i1} + \ldots + t_{im} + t_j$$

- To minimize this increase, we need only to minimize $t_j$. This suggests a simple greedy algorithm: at each step, add to the end of schedule the customer requiring the least service among those remaining.

# Optimality Proof (I)

- **This greedy algorithm is always optimal.**

**(Proof) Let $I = (i_1, \ldots, i_n)$ be any permutation of the integers {1, 2, …, n}. If customers are served in the order $I$, the total time passed in the system by all the customers is**

$$T = t_{i1} + (t_{i1} + t_{i2}) + (t_{i1} + t_{i2} + t_{i3}) + \ldots$$
$$= n\, t_{i1} + (n\text{-}1)t_{i2} + (n\text{-}2)\, t_{i3} + \ldots$$
$$= \sum_{k = 1 \text{ to } n} (n - k + 1)\, t_{ik}$$

# Optimality Proof (II)

- Suppose now that $I$ is such that we can find 2 integers $a$ and $b$ with $a < b$ and $t_{ia} > t_{ib}$ : in other words, the $a$th customer is served before the $b$th customer even though $a$ needs more service time than $b$. If we exchange the positions of these two customers, we obtain a new order of service $I'$. (See the Figure 1) This order is preferable because

$$T(I) = (n-a+1)t_{ia} + (n-b+1)t_{ib} + \sum_{k=1 \text{ to } n \,\&\, k \neq a,b} (n - k + 1)\, t_{ik}$$

$$T(I') = (n-a+1)t_{ib} + (n-b+1)t_{ia} + \sum_{k=1 \text{ to } n \,\&\, k \neq a,b} (n - k + 1)\, t_{ik}$$

$$T(I) - T(I') = (n-a+1)(t_{ia} - t_{ib}) + (n-b+1)(t_{ib} - t_{ia})$$

$$= (b-a)(t_{ia} - t_{ib}) > 0$$

# Optimality Proof (III)

- **We can therefore improve any schedule in which a customer is served before someone else who requires less service. The only schedules that remain are those obtained by putting the customers in non-decreasing order of service time. All such schedules are equivalent and thus they're all optimal.**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Service Order** | $1$ | … | $a$ | … | $b$ | … | $n$ |
| **Served Customer** | $i_1$ | … | $i_a$ | … | $i_b$ | … | $i_n$ |
| **Service Duration** | $t_{ia}$ | … | $t_{ia}$ | … | $t_{ib}$ | … | $t_{in}$ |
| **After exchange $i_a$ & $i_b$** | | | | | | | |
| **Service Duration** | $t_{ia}$ | … | $t_{ib}$ | … | $t_{ia}$ | … | $t_{in}$ |
| **Served Customer** | $i_1$ | … | $i_b$ | … | $i_a$ | … | $i_n$ |

Figure 1