ALGORITHMS AND DATA STRUCTURES I

Course 10

Greedy Algorithms

Previous Course

- Elementary sorting methods belong to $O(n^2)$
- Idea to streamline the sorting process:
  - Divide the initial sequence into two subsequences
  - Sort each subsequence
  - Combine sorted subsequences

	Merge Sort	Quick Sort
Divide	Depending on position	Depending on value
Combine	Merge	Concatenate

Today Course

- Technique of locally optimal choice - "greedy algorithms"
  - Structure
  - The basic idea of the locally optimal choice technique
  - Examples
  - Verification of correctness and analysis of efficiency
  - Applications

Motivation

The stock portfolio problem

Input:

- Amount to be invested (available capital): C
- Action set A= {a_1, a_2, …, a_n }; each action is characterized by
  - Cost (c_1, c_2,..., c_n)
  - Profit (p_1, p_2, ..., p_n)

Output:

- Subset of actions S having the property that:
  - The sum of the costs of shares in S does not exceed C
  - The total profit of shares in S is maximum

What could be the stock selection criteria?

Motivation

Activity selection problem

Consider a set A of activities (e.g. exams) that require a resource (e.g. an exam room). Each activity is characterized by a performance interval (start time and end time). The problem arises of selecting a maximal subset of activities that can be performed using a single resource.

Input:

- Start moments: b_1, b_2, ..., b_n
- Final moments: e_1, e_2,..., e_n

Output:

- Subset S of A having the property that:
  - There are no conflicts between activities: for any two activities i and j in S, their running intervals ($[b_i, e_i)$ and $[b_j, e_j)$ are disjoint)
  - The number of elements in S is maximal

What could be the criteria for selecting activities?

Structure

The general structure of a constrained optimization problem is:

To find x in X (the search space) such that :

(i) x satisfies certain restrictions

(ii) x optimizes (minimizes or maximizes) an optimality criterion

Special case:

- X is a finite set – the problem is in the domain of discrete (or combinatorial) optimization
- At first glance such a problem seems simple to solve. However, since the search space has many elements, exhaustive analysis is impractical, so the problem can become difficult.

Constrained optimization

Example 1. We consider a particular case of the stock portfolio problem when each stock has a cost equal to 1

The problem is equivalent to that of determining a subset of given cardinality and maximum sum

Let $A=\{a_1, \ldots, a_n\}$ and $m<=n$

Determine a subset S of A that satisfies :

(i) The number of elements in S is m (restriction of the problem)

(ii) The sum of the elements in S is maximal (optimal criterion)

Remarks

1. The search space is X = the set of all 2^n subsets of A

2. A brute force approach based on generating all subsets and calculating the sum of their elements would have a complexity of the order of $O(n2^n)$.

Example 1

Problem 1 (find subset with maximum sum under given condition)

Determine the subset S of the finite set A that has the property that :

(i) S has m<=card A elements

(ii) The sum of the elements in S is maximal

Example:

Let A={5,1,7,5,4} and m=3.

Which is the solution? In what order should the elements be selected?

The solution is S={ 7, 5, 5 } - the elements are selected in DECREASING order

Example 1

Greedy approach: the largest m elements of A are selected

```
Subset(A[1..n], m) //variant 1
FOR i ← 1,m DO
  k ← i
  FOR j ← i+1,n DO
    IF A[k]<A[j] THEN k ← j ENDIF
  ENDFOR
  IF k<>i THEN A[k] ↔ A[i] ENDIF
  S[i] ← A[i]
ENDFOR
RETURN S[1..m]
```

```
Subset(A[1..n],m) //variant 2
    A[1..n] ← sort_ decreasing(A[1..n])
    FOR i ← 1,m DO
        S[i] ← A[i]
    ENDFOR
RETURN S[1..m]
```

// less efficient than option 1 (if A is not already sorted and if m is small)

Remark. It can be shown that for this problem the "greedy" technique is better

The basic idea of the locally optimal choice technique

The general optimization problem can be formulated in the form:

Let $A=(a_1, \ldots, a_n)$ be a multiset (a collection of elements that are not necessarily distinct). Find $S=(s_1, \ldots, s_k)$, a subset of A such that, S satisfies certain restrictions and optimizes a criterion.

The idea of optimal local search (greedy search) :

- S is built incrementally starting from the first element
- The most promising at that step is selected from A and added to S.
- Once a choice is made, it is irrevocable (cannot go back and replace a component with another value)

The basic idea of the locally optimal choice technique

The general structure of a "greedy" type algorithm:

```
Greedy(A)
  S ← Ø
  WHILE "S is not completed " AND " there are unselected elements in A" DO
        a ← choose (A) // "choose the best element a, available in A"
       IF S U {a} satisfies the constraints of the problem
      THEN S ← S U {a} // " add a to S"
      ENDIF
  ENDWHILE
RETURN S
```

Remark. The main steps in building the solution : initialization , selection, expansion

The basic idea of the locally optimal choice technique

It represents the most important element of "greedy" type algorithms: the selection of the element that is added at each step.

The selection is made based on a criterion that is established according to the specifics of the problem to be solved

The selection criterion is usually based on heuristics (heuristic = technique based more on experience and intuition than on a deep analysis of the problem = the art of discovering new knowledge (cf. DEX))

Exemple 2

Problem 2 (coins problem)

Assume that we have at our disposal an unlimited number of coins with the values: $\{v_1, v_2, \ldots, v_n\}$. Find a way to cover an amount C so that the number of coins used is as small as possible.

Let s be the number of coins of value v selected

Restriction : $s_1 v_1 + \ldots + s_n v_n = C$

Optimum criterion: the number of selected coins $(s_1 + s_2 + \ldots + s_n)$ is as small as possible

Greedy approach: it starts from the coin with the highest value and covers as much of the amount as possible, for the rest of the amount it tries to use the next coin (in descending order of values) …

Exemple 2

```
coins(v[1..n],C)
   v[1..n] ← sort_descending(v[1..n])
   FOR i ← 1,n DO S[i]:=0 ENDFOR
   i ← 1
   WHILE C>0 and i<=n DO
      S[i] ← C DIV v[i] // maximum number of coins of value v[i]
      C ← C MOD v[i] // the rest remains to be covered
      i ← i+1
   ENDWHILE
   IF C=0 THEN RETURN S[1..n]
   ELSE RETURN "the problem has no solution "
  ENDIF
```

Exemple 2

Remarks :

1. Sometimes the problem has no solution:

   Example: V=(20,10,5) and C=17

   However, if coins of value 1 are available, then the problem always has a solution

2. Sometimes the "greedy" technique does not lead to an optimal solution

   Example : V=(25, 20, 10, 5, 1), C=40

   Greedy approach : (1, 0, 1, 1, 0)

   The optimal solution : (0, 2, 0, 0, 0)

   One condition for optimality : $v_1 > v_2 > \ldots > v_n = 1$ and $v_{i-1} = d_{i-1} v_i$ (i =2 ..n)

Characteristics of the technique

- It does not always lead to a solution optimal (optimal local choice may be have global negative effects; what seems promising at a certain step is possible prove not to be globally optimal)

- Sometimes the solutions obtained by the greedy technique are sub-optimal that is the value CRITERIA is "enough" close to that optimal (an algorithm that leads to sub-optimal solutions it is called approximation algorithm)

Because the "greedy" technique does not guarantee optimal solutions, it is necessary to verify the optimality of the solution for each case (instance).

Correctness check

Many of the problems for which the greedy solution is optimal are characterized by the following properties:

- The optimal substructure property
  - An optimal solution of the initial problem contains an optimal solution of a subproblem (problem of the same type but of smaller size)
- Property of greedy choice
  - The components of an optimal solution have been chosen using the greedy selection criterion or can be replaced by elements chosen using this criterion without altering the optimality property

The optimal substructure property

- When can a problem be said to have the optimal substructure property ?
  - When for an optimal solution $S=(s_1 ,..., s_k)$ of the problem of dimension n, the subset $S_{(2)} = (s_2, ..., s_k)$ is an optimal solution of a subproblem of size (n-1).
- How can one check if a problem has this property ?
  - Using proof by reduction to the absurd

The "greedy" choice property

- When a problem has the "greedy" choice property ?
  - When the optimal solution of the problem is either constructed by a "greedy" strategy or can be transformed in another optimal solution built on the basis of the "greedy" strategy
- How can one check whether a problem possesses this property or not ?
  - It is proved that replacing the first element of an optimal solution with an element selected by the "greedy" technique, the solution remains optimal.

Analysis of its effectiveness

- Greedy algorithms are efficient

- The dominant operation is the selection of the elements (if it is necessary to sort the elements of the set A then the sorting operation is the most expensive)

- So the order of complexity of "greedy" algorithms is

  $O(n^2)$ or $O(n\lg n)$ or $O(n)$

  (depending on the nature of the elements in A and the sorting algorithm used)

Applications

Problem 3: knapsack problem

Consider a set of (n) objects and a backpack of given capacity (C). Each object is characterized by size (d) and value or profit (p). It is required to select a subset of objects so that the sum of their dimensions does not exceed the capacity of the backpack and the sum of the values is maximum.

- Variations:
  - Fractional variant : both whole objects and fractions of objects can be selected. The solution will consist of values belonging to [0,1].
  - Discrete variant (0-1): objects cannot be fragmented, they can only be included in the backpack as a whole.

Knapsack problem

Motivation

Many practical problems are similar to the knapsack problem

Example. Building the financial portfolio: a set of "financial operations"/actions is considered, each one being characterized by a cost and a profit; it is desired to select actions whose total cost does not exceed the amount available for investments and for which the profit is maximum

Example. The knapsack problem (the discrete variant) has applications in cryptography (it was the basis for the development of a public key encryption algorithm – nowadays it is no longer used because it is not secure enough).

Knapsack problem

Example

Val(p)	Size(p)
6	2
5	1
12	3

C=5

Selection criteria

- In ascending order of size (select as many objects as possible)

5+6+12*2/3=11+8=19

Knapsack problem

Example

Val(p)	Size(d)
6	2
5	1
12	3

C=5

Selection criteria

- In ascending order of size (select as many objects as possible)

  5+6+12*2/3=11+8=19

- In decreasing order of value (select the most valuable objects):

  12+6=18

Knapsack problem

Example

Val(p)	Size(p)	Relative profit (Val/Size)
6	2	3
5	1	5
12	3	4

C=5

Selection criteria

- In ascending order of size (select as many objects as possible)

  5+6+12*2/3=11+8=19

- In decreasing order of value (select the most valuable objects):

  12+6=18

- In descending order of relative profit (select the small and valuable items):

  5+12+6*1/2=17+3=20

Knapsack problem

```
Knapsack(d[1..n],p[1..n])
  "sort item (d and p) in descending order by value of relative profit(d/p) "
  FOR i ← 1,n DO S[i] ← 0 ENDFOR
  i ← 1
  WHILE C>0 AND i<=n DO
    IF C>=d[i] THEN S[i] ← 1; C ← C - d[i]
    ELSE S[i] ← C / d[i]; C ← 0
    ENDIF
    i ← i+1
  ENDWHILE
RETURN S[1..n]
```

Knapsack problem

Correctness check:

In the case of the continuous (fractional) version of the problem, the greedy technique leads to the optimal solution

Remark:

- A greedy solution satisfies : S=(1,1,...,1,f,0,..,0)
- $s_1 d_1 + \ldots + s_n d_n = C$ (the restriction can always be satisfied with equality)
- The objects are sorted in descending order according to the value of the relative profit: $p_1 / d_1 > p_2 / d_2 > \ldots > p_n / d_n$

Demo

Let $O=(o_1, o_2, \ldots, o_n)$ be an optimal solution. It can be proven by reduction to the absurd that it is a greedy solution. Lets assume that O is not a greedy solution and consider it as a greedy solution $O'=(o'_1, o'_2, \ldots, o'_n)$

Knapsack problem

Let $B_+ = \{i | o'_i >= o_i\}$ and $B\_ = \{j | o'_j < o_j\}$, k – the smallest index for which $o'_k < o_k$.

Due to the structure of a greedy solution, it follows that any index i in B_+ is smaller than any index j in B_.

On the other hand, both solutions must satisfy the restriction, that is: $o_1 d_1 + \ldots + o_n d_n = o'_1 d_1 + \ldots + o'_n d_n$

$$\sum_{i \in B_+} (o'_i - o_i) d_i = \sum_{i \in B_-} (o_i - o'_i) d_i$$

$$P' - P = \sum_{i=1}^{n} (o'_i - o_i) d_i = \sum_{i \in B_+} (o'_i - o_i) d_i \frac{p_i}{d_i} - \sum_{i \in B_-} (o_i - o'_i) d_i \frac{p_i}{d_i}$$

$$P' - P \geq \frac{p_k}{d_k} \sum_{i \in B_+} (o'_i - o_i) d_i - \frac{p_k}{d_k} \sum_{i \in B_-} (o_i - o'_i) d_i = 0$$

So the greedy solution is at least as good as O (which is a solution optimum). The optimal substructure property is easy to prove by reduction to the absurd.

Knapsack problem

Remark : the result is not valid for the discrete version of the problem

Counterexample

	Val(p)	Size(p)	Relative profit
O1	10	6	5/3
O2	7	5	7/5
O3	6	4	3/2
O4	2	1	2

C=9

- Optimal solution: o2,o3 (total value: 13).
- Solution: dynamic programming technique

Selection criteria

- In ascending order of size (select as many objects as possible)

  o3,o4 (total value: 8)

- In decreasing order of value (select the most valuable objects):

  o1,o4 (total value: 12)

- In descending order of relative profit (select the small and valuable items):

  o4, o1 (total value: 12)

Applications

Problem 4: The task selection problem

Let A={a_1 , ... ,a_n } be a set of activities that share that resource. Each activity requires a time interval to be executed. Two activities are considered compatible if their execution intervals are disjoint and incompatible otherwise.

The problem asks to select as many compatible activities as possible.

Example:

A1: [0,6)

A2: [1,5)

A3: [4,6)

A4: [5,8)

The task selection problem

Let A={a_1 , … ,a_n } be a set of activities that share that resource. Each activity requires a time interval to be executed. Two activities are considered compatible if their execution intervals are disjoint and incompatible otherwise.

The problem asks to select as many compatible activities as possible.

Example:

A1: [0,6)

A2: [1,5)

A3: [4,6)

A4: [5,8)

There are several criteria for selecting activities:

a) In ascending order of start time: a1
b) In ascending order of duration: a3
c) In ascending order of the moment of completion: a2, a4

The task selection problem

```
// Assume that each element of a [ i ] contains two fields:  a[i]. b - beginning moment (begin) and a[i]. e - moment of completion (end)
Select_activities(a[1..n])
  a[1..n] ← ascending sort by the value of e (a[1..n])
  s [1] ← 1 // s will contain indices of the elements selected from a
  k ← 1
  FOR i ← 2,n DO
    IF a[i].b>=a[s[k]].e THEN
      k ← k+1
      s[k] ← i
    ENDIF
  ENDFOR
RETURN s [1..k]
```

The task selection problem

Correctness check. Supposing that the set of activities is ordered increasing by the time of completion ($a_1 e < a_2 e < \ldots < a_k e$).

The "greedy" choice property: Let $(o_1, o_2, \ldots, o_k) = (a_{i1}, a_{i2}, \ldots, a_{ik})$ be an optimal solution (supposing that the selected activities are specified in in ascending order of completion time: $i_1 < i_2 < \ldots < i_k$).

In this case the activity of i_1 can be replaced by a_1, the activity that finishes the fastest, without altering the problem restriction (the selected activities are all compatible) and keeping that and number (maximum) of selected activities

The task selection problem

Correctness check. Supposing that the set of activities is ordered increasing by the time of completion (a_1e < a_2 e <...<a_k e).

The optimal substructure property. Consider the optimal solution : (a_1 ,o_2,...,o_k) (note: from the "greedy" choice property it follows that we can consider o_1 =a_1) .

Supposing that (o_2,...,o_k) is not an optimal solution for the subproblem of selection from {a_2, a_3 ,...,a_n }.

It follows that there is o ' = (o'$_2$,...,o'$_{k'}$) another solution with k ' >k. This would lead to a solution (a_1, o'$_2$,..., o'$_{k'}$) better than (a_1, o_2 ,..., o_k) .

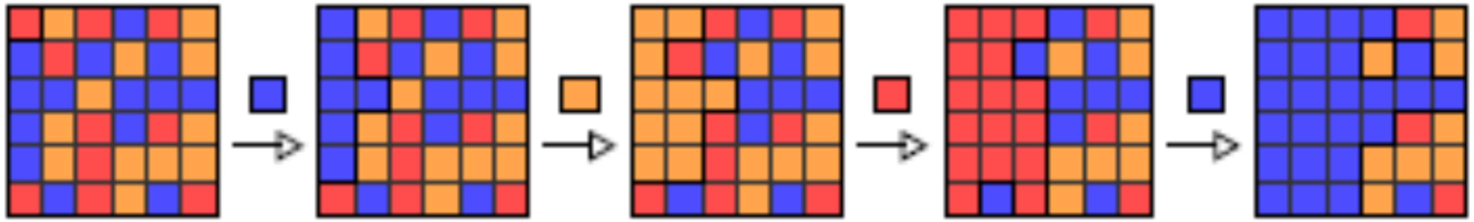
Contradiction

Next course

- Dynamic programming

FloodIt Game [2006, LabPixies->Google]

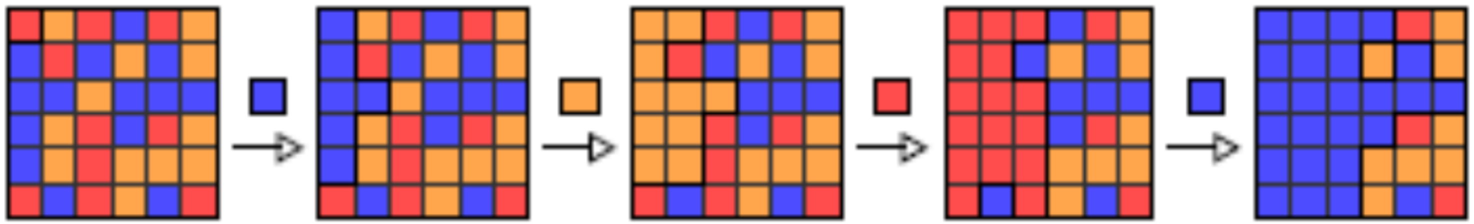
- Consider a nxn grid containing initially randomly colored cells
- The cell in the upper left corner is considered the starting cell
- All cells that have the same color as the start cell and can be accessed by moving horizontally or vertically (but not diagonally) are considered connected to the start cell
- The problem is to successively change the color of the starting cell (and all those connected to it) so that the grid is completely covered with the same color in as few steps as possible



(R. Clifford, The Complexity of Flood Filling Games , 2011)

FloodIt Game [2006, LabPixies->Google]

- Consider a nxn grid containing initially randomly colored cells
- The cell in the upper left corner is considered the starting cell
- All cells that have the same color as the start cell and can be accessed by moving horizontally or vertically (but not diagonally) are considered connected to the start cell
- The problem is to successively change the color of the starting cell (and all those connected to it) so that the grid is completely covered with the same color in as few steps as possible



(R. Clifford, The Complexity of Flood Filling Games , 2011)

What would be a simple/intuitive game strategy?

Q&A