CSE 417: Algorithms and Computational Complexity

5: Dynamic Programming, II Linear Partition

Winter 2002

W. L. Ruzzo

1

Dynamic Programming

- Useful when
  - same recursive sub-problems occur repeatedly
  - Can anticipate the parameters of these recursive calls
  - The solution to whole problem can be figured out with knowing the internal details of how the sub-problems are solved
    - principle of optimality

2

List partition problem

- Given: a sequence of n positive integers $s_1,\ldots,s_n$ and a positive integer k
- Find: a partition of the list into up to k blocks:
  $s_1,\ldots,s_{i_1}|s_{i_1+1}\ldots s_{i_2}|s_{i_2+1}\ldots s_{i_{k-1}}\ |s_{i_{k-1}+1}\ldots s_n$
  so that the sum of the numbers in the largest block is as small as possible.
  i.e. find spots for up to k-1 dividers

3

Greedy approach

- Ideal size would be $P= \sum_{i=1}^{n} s_i/k$
- Greedy: walk along until what you have so far adds up to P then insert a divider
- Problem: it may not exact (or correct)

  100 200 400 500 900 700 600 800 600

  - sum is 4800 so size must be at least 1600.
  - Greedy? Best?

4

Recursive solution

- Try all possible values for the position of the last divider
- For each position of this last divider
  - there are k-2 other dividers that must divide the list of numbers prior to the last divider as evenly as possible
    - $s_1,\ldots,s_{i_1}|s_{i_1+1}\ldots s_{i_2}|s_{i_2+1}\ldots s_{i_{k-1}}\ |s_{i_{k-1}+1}\ldots s_n$
  - recursive sub-problem of the same type

5

Recursive idea

- Let $M[n,k]$ the smallest cost (size of largest block) of any partition of the n into k pieces.
- If best position for last divider lies between the i^{th} and $i+1^{st}$ then

  $$M[n,k]= \max(\underbrace{M[i,k-1]}_{\text{max cost of 1st k-1 blocks}},\ \underbrace{\sum_{j=i+1}^{n} s_j}_{\text{cost of last block}})$$
- In general

  $$M[n,k]= \min_{i<n} \max(M[i,k-1],\ \sum_{j=i+1}^{n} s_j)$$
- Base case(s)?

6

Time-saving - prefix sums

- Computing the costs of the blocks may be expensive and involved repeated work
- Idea: Pre-compute prefix sums
- Length of block

  $s_{i+1}+\ldots+s_j$

  is just

  p[j]-p[i]

- Cost: n additions

$p[1]=s_1$
$p[2]=s_1+s_2$
$p[3]=s_1+s_2+s_3$
...
$p[n]=s_1+s_2+\ldots+s_n$

Linear Partition Algorithm

Partition(S,k):
  p[0]←0;
  for i=1 **to** n **do** p[i] ←p[i-1]+s_i
  for i=1 **to** n **do** M[i,1] ←p[i]
  for j=1 **to** k **do** M[1,j] ← s_1
  for i=2 **to** n **do**
    for j=2 **to** k **do**
      M[i,j] ← $\min_{pos<i}$ {max(M[pos,j-1], p[i]-p[pos])}
      D[i,j] ← value of pos where min is achieved

Linear Partition Algorithm

Partition(S,k):
  p[0]←0; **for** i=1 **to** n **do** p[i] ←p[i-1]+s_i
  for i=1 **to** n **do** M[i,1] ←p[i]
  for j=1 **to** k **do** M[1,j] ← s_1
  for i=2 **to** n **do**
    for j=2 **to** k **do**
      M[i,j]←∞
      for pos=1 **to** i-1 **do**
        s←max(M[pos,j-1], p[i]-p[pos])
        if M[i,j]>s **then**
          M[i,j] ←s ; D[i,j] ←pos

Example:

	1	2	3
100			
200			
400			
500			
900			
700			
600			
800			
600			

Example:

	1	2	3
100	100	100	100
200	300		
400	700		
500	1200		
900	2100		
700	2800		
600	3400		
800	4200		
600	4800		

Example:

	1	2	3
100	100	100	100
200	300	200	200
400	700	400	400
500	1200	700	500
900	2100	1200	900
700	2800	1600	1400
600	3400	2100	
800	4200	2100	
600	4800	2700	