Algorithms Exam [1]

Oct. 19, 2010
kl 14 - 18
byggnad M

Ansvarig:
Devdatt Dubhashi Tel. 073 932 2226 Rum 6479 EDIT

Points :	60	
Grades:	Chalmers	5:48, 4:36, 3:28
	GU	VG:48, G:24
	PhD students	G:36
Helping material :	course textbook, notes	

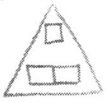
- Recommended: First look through all questions and make sure that you understand them properly. In case of doubt, do not hesitate to ask.
- **Answer all questions in the given space on the question paper (the stapled sheets of paper you are looking at). The question paper will be collected from you after the exam. Only the solutions written in the space provided on the question paper will count for your points.**
- Use extra sheets only for your own rough work and then write the final answers on the question paper.
- Try to give the most effciient solution you can for each problem - your solution will be graded on the basis of its corectness *and* efficiency. In particular a brute force solution will not get any credit.
- Answer concisely and to the point. (English if you can and Swedish if you must!) **Your solution will be graded both for correctness and efficiency - a faster algorithm will get more credit than a slower one.**
- Code strictly forbidden! Motivated pseudocode or plain but clear English/Swedish description is fine.

Lycka till!

[1] 2010 LP 1, INN200 (GU) / TIN090 (CTH).

Problem 1 Median [10] The input consists of two arrays $A[1 \ldots n]$ and $B[1 \ldots n+1]$ containing positive integers, all distinct and both in sorted order. Give a fast algorithm to find the median of all the $2n+1$ numbers i.e. a value $A[i]$ or $B[j]$ such that exactly half the numbers are less than this value and half greater. For example, if $A = [3, 12, 14, 44]$ and $B = [5, 17, 28, 31, 40]$, then the median is $17 = B[2]$. For full credit, your algorithm must run in time $O(\log n)$.

Base case: $n = 1$. Return median of 3 values. Function "med"



A

B

Inductive: cut off appropriate $\lceil n/2 \rceil$ from A, B. proceed on remainder of A, B.

```
proc median(a,b,n) {
    if      n<1 {return ERROR}                                    // sanity
    elseif n=1  {return med(A[a], B[b], B[b+1])}                  // base
    else {                                                        // ind.
        n↑ = ⌈n/2⌉;  n↓ = ⌊n/2⌋;
        if A[a+n↑] < B[b+n↑] { // median in left A or right B
            return median(a, b+n↑, n↓);
        }
        else { // A[a+n↑] > B[b+n↑] : median in right A or left B
            return median(a+n↑, b, n↓);
        }
    }
} // call with median(1,1,n).
```

Constant #op/iteration. 3rd param indicates problem size. 3rd param halves each iter. $\Rightarrow O(\log n)$

Problem 2 Bottlenecks [10] Consider a network of computers represented by a graph $G = (V, E)$: the vertices are computers and an edge represents a communication link between the two endpoints. Each edge e has a number c_e associated with it which is the maximum rate of data transmission it can support. You need to send data form your computer to your friend's computer at the maximum possible rate. If P is a path in G between the vertex u representing your computer and the vertex v representing your friend's, then the maximum rate of sending data along P is determined by the minimum rate c_e of an edge on the path P: this is called the *bottleneck* rate of the path P. Thus you want to find a path P between u and v with maximum bottleneck rate.

(a) Give a greedy algorithm to solve the problem. [4 pts]

Modify Dijkstra's algorithm s.t. d(w) is bottleneck between u and w (instead of min. cost)

```
proc path(G, u, v) { // G = (V,E)
    S := {u};
    For each w∈V {d(w):=0};
    while S ≠ V {
        d' = 0; W_next = null; W_cur = null;
        For each e = (w,w') ∈ E ∩ S×(V\S) {
            t = min(d(w), c_e);
            if t > d' {
                d' = t; W_next = w'; W_curr = w;
            }
        }
        S := S ∪ {w_next}; d(w_next) := d'; p(w_next) := W_curr;
    }
    if d(u) = 0 {return NO-PATH}
    else return p; // p is a "predecessor in cheapest path" function.
}
```

greediness

```
proc P(p, u, w){
    P_w = [w];
    while u ≠ w {
        w := p(w); P_w := w::P_w;
    } return P_w
}
```

Let $P_w = P(p, u, w)$, where $p = path(G, u, v)$.

(b) Argue that your algorithm is correct and optimal. [3pts]

Same argument as in Dijkstra's algorithm.

Claim: For S, p at any point in exec. of path(G,u,v), For each $w \in S$, P_w is best (max bottleneck) u-w path in G.

proof: induction in $|S| = n$.

base: $n=1$, $S = \{u\}$, only w in S is u. $P_u = [u]$.

ind: IH: $P_{w'}$ is best u-w path in G, for each $w' \in S'$.

$S = S' \cup \{w\}$. P_w is best way to reach w from S' in 1 step. Assume some P_w', which detours through some $\hat{w} \in V \setminus S'$, out of S', into w, is better. Algorithm picked w since w has best bottleneck from S'. Path through $\hat{w}$ thus at most as good as P_w; contradiction.

all u-w paths must leave S' somewhere.

(c) What is the running time of your algorithm? Justify with the appropriate data structures. [3 pts]

Computing P(p,u,v) from p = path(G,u,v): $O(|V|)$.

By not discarding d' and using priority queue for storing $(d'(w), w)$ for $w \in V \setminus S$, we can obtain $O(|E| \log |V| + |V|) = O(|V|^2 \log |V|)$.

PROBLEM 3, greedy solution sketch

```
proc maxbiton(A) {
    if n=0 {return 0}
    else {
                                    ( no pun intended :-)
                                    BS
        l := 0 ;          //max length bitonic subsequence
        cnt:= 1;          // size of current candidate
        down = true;      // we are in a downward slope in

        for i = 2..n {
            if A[i-1] > A[i] {
                cnt++;
                down = true;
            } else // A[i-1] < A[i]  // up
                if down { //we bottomed; new BS starts
                    down:= False;
                    l := max{l, cnt};
                    cnt := 2;    // A[i-1], A[i]
                } else           // still going up
                    cnt++;
                }
            }
        }
        l := max{l, cnt};
        return l;
    }
}
```

PROBLEM 3, D&C SOLUTION Sketch

```
Srise (A){   ~ 3-5 ops
    if |A|=0 {return false}
    elseif |A|=1 {return true}
    else return A[1] < A[2]
}

Sfall (A){
    if |A|=0 {return false}
    elseif |A|=1 {return true}
    else return A[1] > A[2]
}

Erise (A){
    if |A|=0 {return false}
    elseif |A|=1 {return true}
    else return A[|A|-1] < A[|A|]
}

Efall (A){
    if |A|=0 {return false}
    elseif |A|=1 {return true}
    else return A[|A|-1] > A[|A|]
}
```

Divide: Split A in "half", L and R.
Remember where split occurred in A. ⊛
Call recursively on L and R.

Conquer: Get back L' and R'.

```
if adjacent in A { // check uses ⊛.        ⇒ legal to combine
    Le := last elem. in L'
    Rs := first elem. in R'
    if ( Erise(L') ^ Srise(R') ^ Le < Lb      //
      or
      Efall(L') ^ Sfall(R') ^ Le > Lb         //
      or
      Erise(L') ^ Sfall(R') ) {               //
        return L'R'
    } else  // cannot combine
        return longest(L',R')
    }
} else {
    return longest(L',R')
}
```

assumes |L'| takes $O(1)$ to compute, for instance

$T(n) \le 2T(\frac{n}{2}) + 24$ ~ conquer

$T(1) = 1$ ~ return cell

$\cancel{2(2T(\frac{n}{4}) + 24) + 24}$

$\cancel{= 2(2T(\frac{n}{4})) + 3 \cdot 24}$

Problem 4 Planning Cycle Trip [10] You are planning a cycle trip in the summer along the east coast starting at Stockholm and ending in Kiruna. You begin on day 1 at Stockholm and on day k you arrive at Kiruna, in between there are n towns numbered $1..n$ where you can rest for the night. Let us number Stockholm as 0, Kiruna as $n+1$. There is an array $d[1 \ldots n+1]$ such that the distance from Stockholm to city i is given by $d[i]$ and so the distance between succesive towns i and $i+1$ is given $d[i+1] - d[i]$. To rest for the night at the only "vandrahem" (youth hostel) in city i costs $c[i]$, for $i = 1 \cdots n$. Assume $c[0] = 0 = c[n+1]$. The total distance to cycle is $d[n+1]$ and you would like to spread this as evenly as possible between the k days, so that on average you cycle $\bar{d} := d[n+1]/k$ each day. However, this may not always be possible and sometimes you cycle a bit more and sometimes a bit less. If you cycle y km on a certain day, you penalty for this day is $(y - \bar{d})^2$. Thus if you cycle y_i km on day i and stay the night at vandrahem in city v_i, then your total cost is

$$\sum_{1 \leq i \leq k} \{c[v_i] + (y_i - \bar{d})^2\}.$$

Your goal is to find a schedule to cycle that minimizes this total cost.
In this problem, you develop a dynamic programing solution to the problem. Let $OPT(i,j)$ be the minimum cost for ending at town i on day j.

(a) [1 pt] In this notation, what is the final solution we want?

$OPT(n+1, k)$ (sleeping @ home is free)

(b) [1 pt] What is the value of $OPT(0,j)$? $OPT(i,0)$?

$OPT(0,j) = \sum_{1 \leq k \leq j} 0 + (0-\bar{d})^2 = j\bar{d}^2$ $\quad OPT(i,0) = \infty$ (must be in Stockholm @ day 1)

(c) [3 pts] Write a recurrence for $OPT(i,j)$. (HINT: consider what is done on day j.)

$$OPT(i,j) = \min_{0 \leq \hat{i} < i} \{c[i] + (d[i] - d[\hat{i}] - \bar{d})^2 + OPT(\hat{i}, j-1)\}$$

(with $d[0] := 0$)

(d) [2 pts] Using (b) and (c), implement the recurrence efficiently in pseudocode.

```
for i in 0..n+1 { OPT[i,0] := ∞ }
for j in 0..k { OPT[0,j] := j·d̄² }  // OPT[0,0] = 0
for j in 1..k {
    for i = 1..n+1 {
        m := ∞
        for î = 0..i {
            m := min(m, c[i]+(d[i]-d[î]-d̄)²+OPT[î, j-1])
        }
        OPT[i,j] := m
    }
}
return OPT[n+1,k]
```

Problem 6 hint

CLIQUE $\leq_p$ PROF