Dynamic Programming

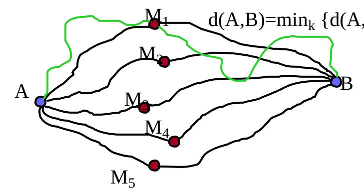
EECS 477

Lecture 15, 11/5/2002

Dynamic Programming

- Solution splits into parts
- If a solution is optimal then its parts have to be optimal too

$d(A,B)=\min_k \{d(A,M_k) + d(M_k,B)\}$



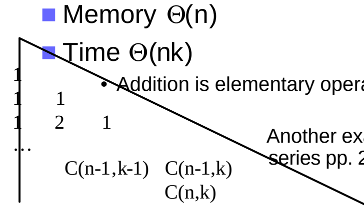
Algorithm

- Compute subparts for smaller instances, store results
- Combine them
- Bottom-up approach
- Simple example: C(n,k)

```
unsigned Binomial(unsigned n, unsigned k) {
    if(k==0 || k==n) return 1;
    else return Binomial(n-1,k-1) + Binomial(n-1,k);
} //// Ω(C(n,k)) algorithm
```

Pascal's triangle

- Keep intermediate results
  - Just one line of the table suffices
- Memory $\Theta(n)$
- Time $\Theta(nk)$
  - Addition is elementary operation



Another example: World series pp. 261-262

Making change

- Pay a given amount with smallest number of coins
  - Greedy algorithm doesn't always work
    - {1,4,6} paying 8: greedy 6+1+1, optimal 4+4
  - Paying out 15: 6+4+4+1 is optimal
    - Subparts are optimal too 10=6+4 and 5=4+1
    - Once we know how to pay 10 optimally we should remember that: *build a table*

Making change: pay amount N

- Coins
  - Denominations d[1],…,d[M]
- Table c(i,j): i=1..M, j=0..N
  - the minimum number of coins to pay amount j using coins d[1],…,d[i]
- Optimality

  c(i,0) = 0

  c(i,j) = min { c(i-1,j), 1+c(i,j-d[i]) }

  If any value falls outside of the table put it to $+\infty$

Making change

- c(i,j) = min { c(i-1,j), 1+c(i,j-d[i]) }
- Coins: {2,3,7}
  - Fill by rows, read off solution later

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
2	0	Inf	1	Inf	2	Inf	3	Inf	4	Inf	5	Inf	6	Inf
3	0	Inf	1	1	2	2	2	3	3	3	4	4	4	5
7	0	Inf	1	1	2	2	2	1	3	2	2	3	3	3

Making change

- Runtime to fill the table: Θ((N+1)*M)
- Runtime to extract the set of coins
  M steps up, c(M,N) steps left.
  Total: Θ(M+c(M,N))
- What is different between this and D&C approach?
  - List of things

Knapsack problem

- Non-breakable objects: i=1..N
- Weights w_i, value v_i
- Now we can x_i=0 or 1
- Constraint $\Sigma_i x_i w_i \leq W$
- Maximize $\Sigma_i x_i v_i$
- Greedy no longer works: W = 5
  { (4oz, $28), (3oz, $18), (2oz, $12) }

Knapsack: DP

- V[i,j] = maximum value if W=j and we can choose among objects 1..i
- V[i,j] = max{ V[i-1,j], V[i-1, j-w_i]+v_i }
- V[0,j] = 0, when j≥0
- V[i,j] = -∞, when j<0
- V[i,0] = 0
- Build a table again

Knapsack: table

- W = 12
  - fill by row

	0	1	2	3	4	5	6	7	8	9	10	11	12
2oz, $4	0	0	4	4	4	4	4	4	4	4	4	4	4
3oz, $7	0	0	4	7									
5oz, $2	0	0											
7oz, $6	0	0											

Knapsack

- Algorithm
- Runtime Θ(nW)
- Finding the load composition O(n+W)
- Is this fast or slow?
- What would be a bad example?

Floyd's algorithm

- Shortest paths in a directed graphs between all the pairs of vertices
  - Dijkstra does paths from one seed vertex
- Graph G=[N={1,..,N} ,A]
  - Arrows A – stored in the edge length matrix
  L[i,j] = distance from i to j, infinity if no edge
- If k is on the shortest path from i to j, then (i to k), and (k to j) is optimal too

Floyd's algorithm

- Constructing matrix D of shortest path distances
- D_k is the matrix of shortest paths using only vertices 1..k as intermediate
- $D_k[i,j] = \min \{ D_{k-1}[i,j], D_{k-1}[i,k]+D_{k-1}[k,j] \}$
- Start with $D_0 = L$
- N by N matrix N times
  - Runtime $\Theta(N^3)$ (Dijkstra $N\Theta((A+N) \log N)$)

Chained matrix multiplication

- $c_{ij} = \Sigma_k a_{ik} b_{kj}$
- A – p by q matrix
- B – q by r matrix
- AB takes pqr scalar multiplication
- Example ABCD: what is the best order?
- 2x3, 3x5, 5x2, 2x7
- Greedy algorithm does not work

Chained matrix multiplication

- D[0], D[1], …, D[N] dimensions
- Matrix M_i has dimensions D[i-1] x D[i]
- Optimality
  $P(i,i+s) = \min_{i \le k \le i+s} \{P(i,k)+P(k+1,i+s)+D[i-1]D[k]D[i+s]\}$
- Start with P(i,i+1) = D[i-1]D[i]D[i+1]
- Go to higher s