CS 473ug: Algorithms

Chandra Chekuri
chekuri@cs.uiuc.edu
3228 Siebel Center

University of Illinois, Urbana-Champaign

Fall 2007

Part I

Optimal Binary Search Trees

Binary Search Trees

Given n sorted keys/numbers $a_1 < a_2 < \ldots < a_n$.
Data structure to store keys so that one can answer dictionary query: is a one of the keys?

Binary Search Trees

Given n sorted keys/numbers $a_1 < a_2 < \ldots < a_n$.
Data structure to store keys so that one can answer dictionary query: is a one of the keys?

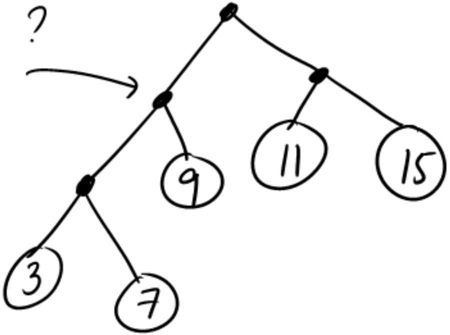
Binary Search Tree:

- a full binary tree T
- keys at leaves of the tree
- leaves in left to right give sorted order $a_1, a_2, \ldots, a_n$
- internal node stores relevant information to guide search

Given a, can walk down the tree to check if a in the tree or not.

Example

3, 7, 9, 11, 15



Balanced Binary Search Trees

General setting: keys are dynamic with insertions, deletions, etc.

Dynamic search trees: keep tree balanced so that height of tree is $O(\log n)$. Search/insertion/deletion take $O(\log n)$ time.

Static Setting with Statistical Information

Static setting:

- keys $a_1, a_2, \ldots, a_n$ known in advance
- no insertions or deletions, only search queries
- also know frequencies of search queries: p_i probability of querying a_i

Static Setting with Statistical Information

Static setting:

- keys $a_1, a_2, \ldots, a_n$ known in advance
- no insertions or deletions, only search queries
- also know frequencies of search queries: p_i probability of querying a_i

Problem: design a binary search tree T so as to minimize the *average* search time

$$\sum_{i=1}^{n} p_i s_T(a_i)$$

where $s_T(a_i)$ is the search time for a_i in T.

Static Setting with Statistical Information

Static setting:

- keys $a_1, a_2, \ldots, a_n$ known in advance
- no insertions or deletions, only search queries
- also know frequencies of search queries: p_i probability of querying a_i

Problem: design a binary search tree T so as to minimize the *average* search time

$$\sum_{i=1}^{n} p_i s_T(a_i)$$

where $s_T(a_i)$ is the search time for a_i in T.

What is $s_T(a_i)$?

Static Setting with Statistical Information

Static setting:

- keys $a_1, a_2, \ldots, a_n$ known in advance
- no insertions or deletions, only search queries
- also know frequencies of search queries: p_i probability of querying a_i

Problem: design a binary search tree T so as to minimize the *average* search time

$$\sum_{i=1}^{n} p_i s_T(a_i)$$

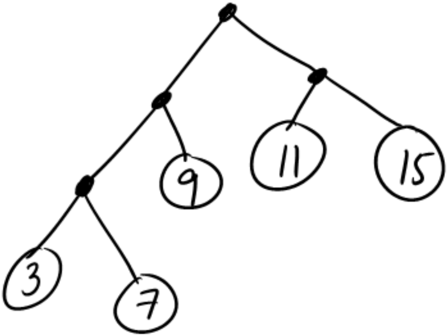
where $s_T(a_i)$ is the search time for a_i in T.

What is $s_T(a_i)$? depth of a_i in T denoted by $d_T(a_i)$

Example

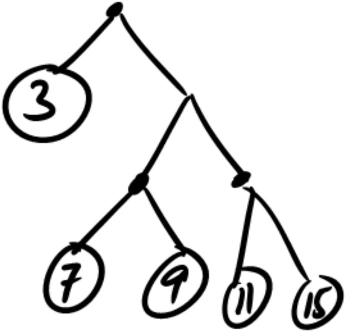
3, 7, 9, 11, 15

$\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}$



3, 7, 9, 11, 15

$0.9, \frac{1}{40}, \frac{1}{40}, \frac{1}{40}, \frac{1}{40}$



Real Problem

Can search for any key a

Statistical information: $q_0, p_1, q_1, p_2, q_2, \ldots, p_n, q_n$

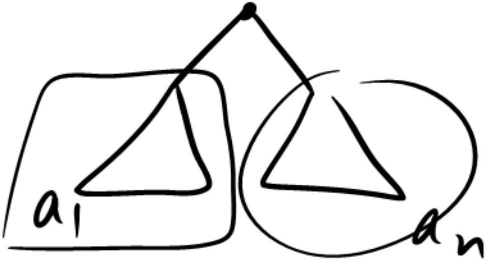
- p_i: probability that a_i is searched for
- q_i: probability that a number a in the range (a_i, a_{i+1}) is searched for

Simpler problem ideas can be extended to the above real problem.

Optimal Binary Search Trees: Recursive Solution?

Can we solve the problem recursively?

$S(i,j)$: optimum cost of a binary search tree for $a_i, a_{i+1}, \ldots, a_j$ with probabilities $p_i, p_{i+1}, \ldots, p_j$
Want $S(1,n)$



Optimal Binary Search Trees: Recursive Solution?

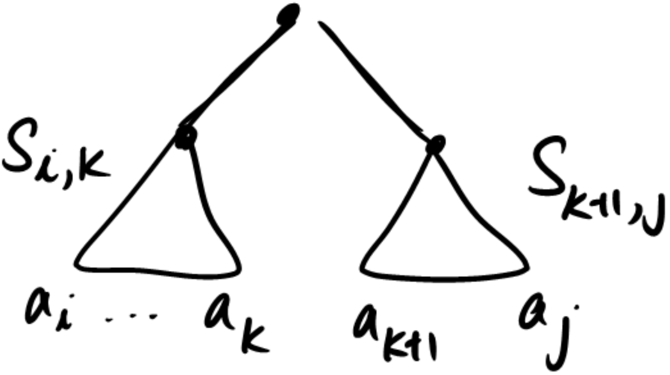
Can we solve the problem recursively?

$S(i,j)$: optimum cost of a binary search tree for $a_i, a_{i+1}, \ldots, a_j$
with probabilities $p_i, p_{i+1}, \ldots, p_j$
Want $S(1,n)$

Recurrence for $S(i,j)$

$$S(i,j) = \min_{i \leq k < j} \left(S(i,k) + S(k+1,j) + \sum_{k=i}^{j} p_k \right)$$

$a_i, a_{i+1}, \ldots, a_j$



Optimal Binary Search Trees: Recursive Solution?

Can we solve the problem recursively?

$S(i,j)$: optimum cost of a binary search tree for $a_i, a_{i+1}, \ldots, a_j$
with probabilities $p_i, p_{i+1}, \ldots, p_j$
Want $S(1,n)$

Recurrence for $S(i,j)$

$$S(i,j) = \min_{i \le k < j} \left(S(i,k) + S(k+1,j) + \sum_{k=i}^{j} p_k \right)$$

Base case: $S(i,i) = p_i$ for $1 \le i \le n$

Iterative Algorithm

$$S(i,j) = \min_{i \le k < j} \left(S(i,k) + S(k+1,j) + \sum_{k=i}^{j} p_k \right)$$

Iterative Algorithm

$$S(i,j) = \min_{i \le k < j} \left(S(i,k) + S(k+1,j) + \sum_{k=i}^{j} p_k \right)$$

Base case: $S(i,i) = p_i$ for $1 \le i \le n$

How many subproblems?

Iterative Algorithm

$$S(i,j) = \min_{i \le k < j} \left(S(i,k) + S(k+1,j) + \sum_{k=i}^{j} p_k \right)$$

Base case: $S(i,i) = p_i$ for $1 \le i \le n$

How many subproblems? $O(n^2)$

Iterative Algorithm

$$S(i,j) = \min_{i \le k < j} \left(S(i,k) + S(k+1,j) + \sum_{k=i}^{j} p_k \right)$$

Base case: $S(i,i) = p_i$ for $1 \le i \le n$

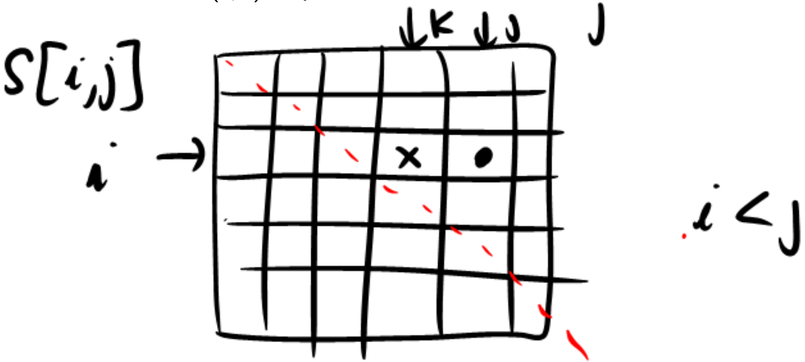
How many subproblems? $O(n^2)$

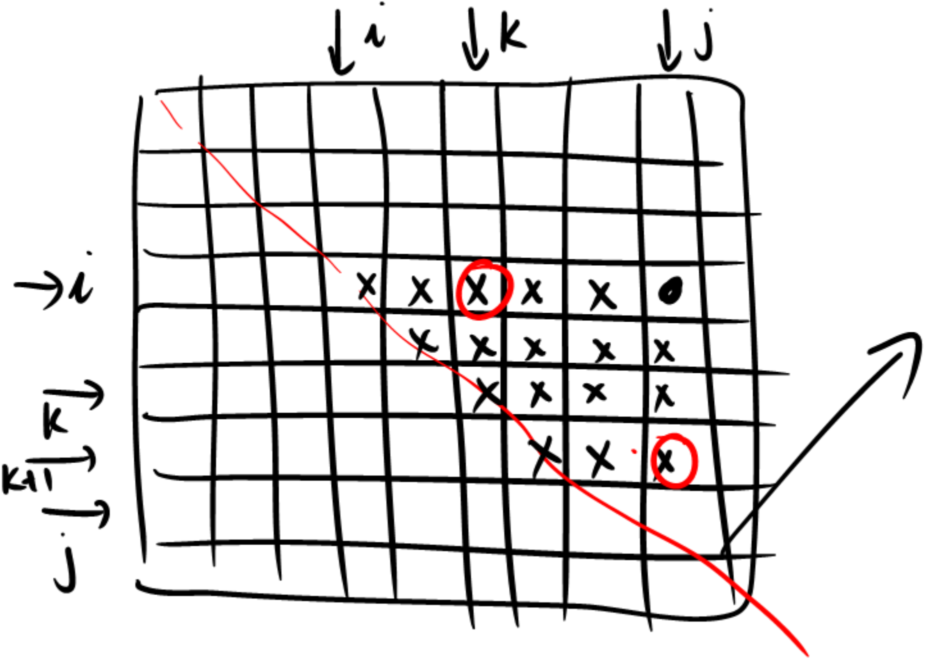
Precomputation: $P(i,j) = \sum_{k=i}^{j} p_k$ in $O(n^2)$ time.

Iterative Algorithm

$$S(i,j) = \min_{i \le k < j} \left(S(i,k) + S(k+1,j) + P(i,j) \right)$$

Base case: $S(i,i) = p_i$ for $1 \le i \le n$





i
k
j
i
k
k+1
j

Iterative Algorithm

$$S(i,j) = \min_{i \le k < j} \left(S(i,k) + S(k+1,j) + P(i,j)\right)$$

Base case: $S(i,i) = p_i$ for $1 \le i \le n$

```
for i = 1 to n do
    S[i,i] = P[i,i]
```

Iterative Algorithm

$$S(i,j) = \min_{i \le k < j} \left(S(i,k) + S(k+1,j) + P(i,j) \right)$$

Base case: $S(i,i) = p_i$ for $1 \le i \le n$

```
for i = 1 to n do
    S[i,i] = P[i,i]

for d = 1 to n - 1 do
    for i = 1 to n - d do
        j = i + d
        S[i,j] = min_{i≤k<j}(S[i,k] + S[k+1,j] + P[i,j])
```

Running time:

Iterative Algorithm

$$S(i,j) = \min_{i \le k < j} (S(i,k) + S(k+1,j) + P(i,j))$$

Base case: $S(i,i) = p_i$ for $1 \le i \le n$

```
for i = 1 to n do
    S[i,i] = P[i,i]

for d = 1 to n − 1 do
    for i = 1 to n − d do
        j = i + d
        S[i,j] = min_{i≤k<j}(S[i,k] + S[k+1,j] + P[i,j])
```

Running time: $O(n^3)$
Space:

Iterative Algorithm

$$S(i,j) = \min_{i \le k < j} (S(i,k) + S(k+1,j) + P(i,j))$$

Base case: $S(i,i) = p_i$ for $1 \le i \le n$

```
for i = 1 to n do
    S[i,i] = P[i,i]

for d = 1 to n − 1 do
    for i = 1 to n − d do
        j = i + d
        S[i,j] = min_{i≤k<j}(S[i,k] + S[k+1,j] + P[i,j])
```

Running time: $O(n^3)$
Space: $O(n^2)$

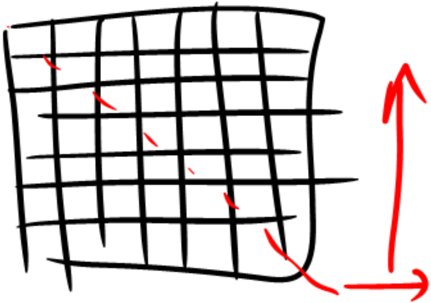
Computing the Table: Alternative 1

```
for i = 1 to n do
    S[i, i] = P[i, i]
```

Computing the Table: Alternative 1

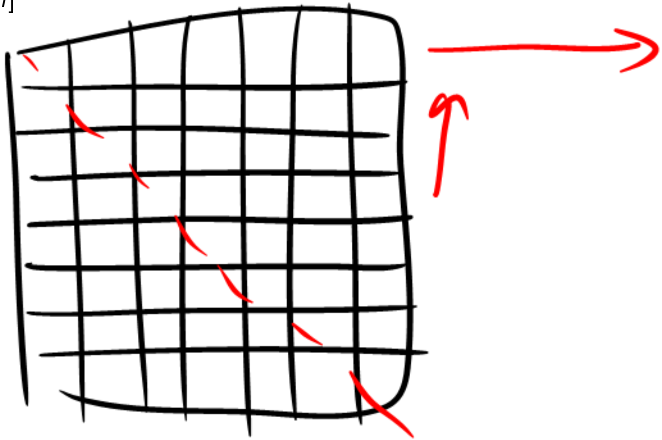
for $i = 1$ to n do
    $S[i, i] = P[i, i]$

for $i = n$ downto 1 do
    for $j = i + 1$ to n do
    $S[i, j] = \min_{i \le k < j}(S[i, k] + S[k + 1, j] + P[i, j])$



Computing the Table: Alternative 2

for $i = 1$ to n do
    $S[i, i] = P[i, i]$



Computing the Table: Alternative 2

```
for i = 1 to n do
    S[i, i] = P[i, i]

for j = 1 to n do
    for i = j − 1 downto 1 do
    S[i, j] = min_{i≤k<j}(S[i, k] + S[k + 1, j] + P[i, j])
```

Part II

Knapsack

Knapsack Problem

Input

- n items. each item i has a positive integer size s_i and a positive integer profit p_i.
- a knapsack of integer capacity B.

Goal Pack a maximum profit subset of items into knapsack.

Example

Item	Size	Profit
1	6	\$30
2	3	\$14
3	4	\$16
4	2	\$9

$B = 10$

Potential Solutions: $\{1,3\}, \{2,3,4\}$

OPT Solution : $\{1,3\}$

Towards a Recursive Solution

Observation

Consider an optimal solution $\mathcal{O}$

Case item $n \in \mathcal{O}$ *Then $\mathcal{O} - \{n\}$ is an optimum solution for items 1 to $n-1$ in knapsack of capacity $B - s_n$*

Case item $n \notin \mathcal{O}$ *$\mathcal{O}$ is an optimal solution to items 1 to $n-1$*

Towards a Recursive Solution

Observation

Consider an optimal solution $\mathcal{O}$

Case item $n \in \mathcal{O}$ *Then $\mathcal{O} - \{n\}$ is an optimum solution for items 1 to $n-1$ in knapsack of capacity $B - s_n$*

Case item $n \notin \mathcal{O}$ *$\mathcal{O}$ is an optimal solution to items 1 to $n-1$*

Subproblems depend also on *remaining* capacity.

$OPT(i, C)$: optimum profit for items 1 to i in knapsack of size C

Goal: compute $OPT(n, B)$

Recursive Solution

$OPT(i, C)$: optimum profit for items 1 to i in knapsack of size C

$$OPT(i, C) = \max \begin{cases} p_i + OPT(i-1, C-s_i) & \text{if } s_i \leq C \\ 0 & \text{if } s_i > C \\ OPT(i-1, C) & \end{cases}$$

Recursive Solution

$OPT(i, C)$: optimum profit for items 1 to i in knapsack of size C

$$OPT(i, C) = \max \begin{cases} p_i + OPT(i-1, C-s_i) & \text{if } s_i \leq C \\ 0 & \text{if } s_i > C \\ OPT(i-1, C) & \end{cases}$$

Base case: $OPT(i, 0) = 0$ for $i = 1$ to n.

Recursive Solution

$OPT(i, C)$: optimum profit for items 1 to i in knapsack of size C

$$OPT(i, C) = \max \begin{cases} p_i + OPT(i-1, C-s_i) & \text{if } s_i \leq C \\ 0 & \text{if } s_i > C \\ OPT(i-1, C) & \end{cases}$$

Base case: $OPT(i, 0) = 0$ for $i = 1$ to n.

How many subproblems?

Recursive Solution

$OPT(i, C)$: optimum profit for items 1 to i in knapsack of size C

$$OPT(i, C) = \max \begin{cases} p_i + OPT(i-1, C-s_i) & \text{if } s_i \leq C \\ 0 & \text{if } s_i > C \\ OPT(i-1, C) & \end{cases}$$

Base case: $OPT(i, 0) = 0$ for $i = 1$ to n.

How many subproblems? $O(nB)$

Iterative Algorithm

```
for i = 0 to n do
    OPT[i, 0] = 0

for i = 1 to n do
    for C = 1 to B do
        if s_i ≤ C then
            OPT[i, C] = max(OPT[i − 1, C], p_i + OPT[i − 1, C − s_i])
        else
            OPT[i, C] = OPT[i − 1, C]

Output OPT[n, B]
```

Running time:

Iterative Algorithm

```
for i = 0 to n do
    OPT[i, 0] = 0

for i = 1 to n do
    for C = 1 to B do
        if s_i ≤ C then
            OPT[i, C] = max(OPT[i − 1, C], p_i + OPT[i − 1, C − s_i])
        else
            OPT[i, C] = OPT[i − 1, C]

Output OPT[n, B]
```

Running time: $O(nB)$
Space:

Iterative Algorithm

```
for i = 0 to n do
    OPT[i, 0] = 0

for i = 1 to n do
    for C = 1 to B do
        if s_i ≤ C then
            OPT[i, C] = max(OPT[i − 1, C], p_i + OPT[i − 1, C − s_i])
        else
            OPT[i, C] = OPT[i − 1, C]

Output OPT[n, B]
```

Running time: $O(nB)$
Space: $O(nB)$

Knapsack Algorithm and Polynomial time

Input size for Knapsack:

Knapsack Algorithm and Polynomial time

Input size for Knapsack: $O(n) + \log B + \sum_{i=1}^{n}(\log s_i + \log p_i)$

Knapsack Algorithm and Polynomial time

Input size for Knapsack: $O(n) + \log B + \sum_{i=1}^{n}(\log s_i + \log p_i)$

Running time of dynamic programming algorithm: $O(nB)$

Not a polynomial time algorithm!

Example: $B = 2^n$ and $s_i, p_i \in [1..2^n]$.
Input size is $O(n)$, running time is $O(n2^n)$.

Algorithm is called a *pseudo-polynomial* time algorithm because running time is polynomial if *numbers* in input are of size polynomial in *combinatorial* size of problem.

Knapsack is NP-hard if numbers are not polynomial in n!

Part III

Traveling Salesman Problem

Traveling Salesman Problem

Input A graph $G = (V, E)$ with non-negative edge costs/lengths. $c(e)$ for edge e

Goal Find a tour of minimum cost that visits each node.

Traveling Salesman Problem

Input A graph $G = (V, E)$ with non-negative edge costs/lengths. $c(e)$ for edge e

Goal Find a tour of minimum cost that visits each node.

No polynomial time algorithm known. Problem is NP-Hard.

Example


An Exponential Time Algorithm

How many different tours are there?

An Exponential Time Algorithm

How many different tours are there? $n!$

An Exponential Time Algorithm

How many different tours are there? $n!$

Stirling's formula: $n! \simeq \sqrt{n}(n/e)^n$

An Exponential Time Algorithm

How many different tours are there? $n!$

Stirling's formula: $n! \simeq \sqrt{n}(n/e)^n$ which is $\Theta(2^{cn \log n})$ for some constant $c > 1$

An Exponential Time Algorithm

How many different tours are there? $n!$

Stirling's formula: $n! \simeq \sqrt{n}(n/e)^n$ which is $\Theta(2^{cn \log n})$ for some constant $c > 1$

Can we do better? Can we get a $2^{O(n)}$ time algorithm?

A More General Path Problem

Given G and nodes v_i, v_j find a minimum cost path from v_i to v_j that visits every node exactly once.

A More General Path Problem

Given G and nodes v_i, v_j find a minimum cost path from v_i to v_j that visits every node exactly once.

Can solve TSP using above. Do you see how?

A More General Path Problem

Given G and nodes v_i, v_j find a minimum cost path from v_i to v_j that visits every node exactly once.

Can solve TSP using above. Do you see how?

Let $f(i, j, V)$ be minimum cost path from v_i to v_j that visits all nodes.
Can we express this as a recursive solution?

A More General Path Problem

Given G and nodes v_i, v_j find a minimum cost path from v_i to v_j that visits every node exactly once.

Can solve TSP using above. Do you see how?

Let $f(i, j, V)$ be minimum cost path from v_i to v_j that visits all nodes.
Can we express this as a recursive solution?

What is the next node in the optimum path from i to j? Suppose it v_k. Then what is $f(i, j)$?

$$f(i, j, V) = c(v_i, v_k) + f(k, j, V - \{i\})$$

A Recursive Solution

$$f(i, j, V) = \min_{k \neq i,j} (c(v_i, v_k) + f(k, j, V - \{i\}))$$

Why is $f(k, j, V - \{i\})$ a subproblem?

A Recursive Solution

$$f(i, j, V) = \min_{k \neq i,j} (c(v_i, v_k) + f(k, j, V - \{i\}))$$

Why is $f(k, j, V - \{i\})$ a subproblem?
What are the subproblems?

A Recursive Solution

$$f(i, j, V) = \min_{k \neq i,j} (c(v_i, v_k) + f(k, j, V - \{i\}))$$

Why is $f(k, j, V - \{i\})$ a subproblem?
What are the subproblems?
$f(a, b, S)$ for $a = 1, 2, \ldots, n$, $b = 1, 2, \ldots, n$, $S \subseteq V$.

How many subproblems?

A Recursive Solution

$$f(i, j, V) = \min_{k \neq i,j} (c(v_i, v_k) + f(k, j, V - \{i\}))$$

Why is $f(k, j, V - \{i\})$ a subproblem?
What are the subproblems?
$f(a, b, S)$ for $a = 1, 2, \ldots, n$, $b = 1, 2, \ldots, n$, $S \subseteq V$.

How many subproblems? $O(n^2 2^n)$
Exercise: Show that one can compute TSP using above dynamic program in $O(n^3 2^n)$ time and $O(n^2 2^n)$ space.

Disadvantage of dynamic programming solution:

A Recursive Solution

$$f(i, j, V) = \min_{k \neq i,j} (c(v_i, v_k) + f(k, j, V - \{i\}))$$

Why is $f(k, j, V - \{i\})$ a subproblem?
What are the subproblems?
$f(a, b, S)$ for $a = 1, 2, \ldots, n$, $b = 1, 2, \ldots, n$, $S \subseteq V$.

How many subproblems? $O(n^2 2^n)$
Exercise: Show that one can compute TSP using above dynamic program in $O(n^3 2^n)$ time and $O(n^2 2^n)$ space.

Disadvantage of dynamic programming solution: memory!