Amortized Analysis

Amortized Analysis

- Recall how we calculate the time complexity of dynamic programming:
  - Count the number n of subproblems.
  - Compute the worst case time complexity t of construct the optimal solution from the optimal solutions of the subproblems.
  - The total time complexity: O(nt)

Amortized Analysis

- The analysis might be inaccurate.
  - Not all subproblems has running time t.
  - Actually, they has running time $\leq t$.
- Idea
  - Compute the average case time complexity t', not the worst case.
  - The total time complexity: $O(nt')$
- Sometimes we may have $t'=o(t)$!

Example: Optimal BST

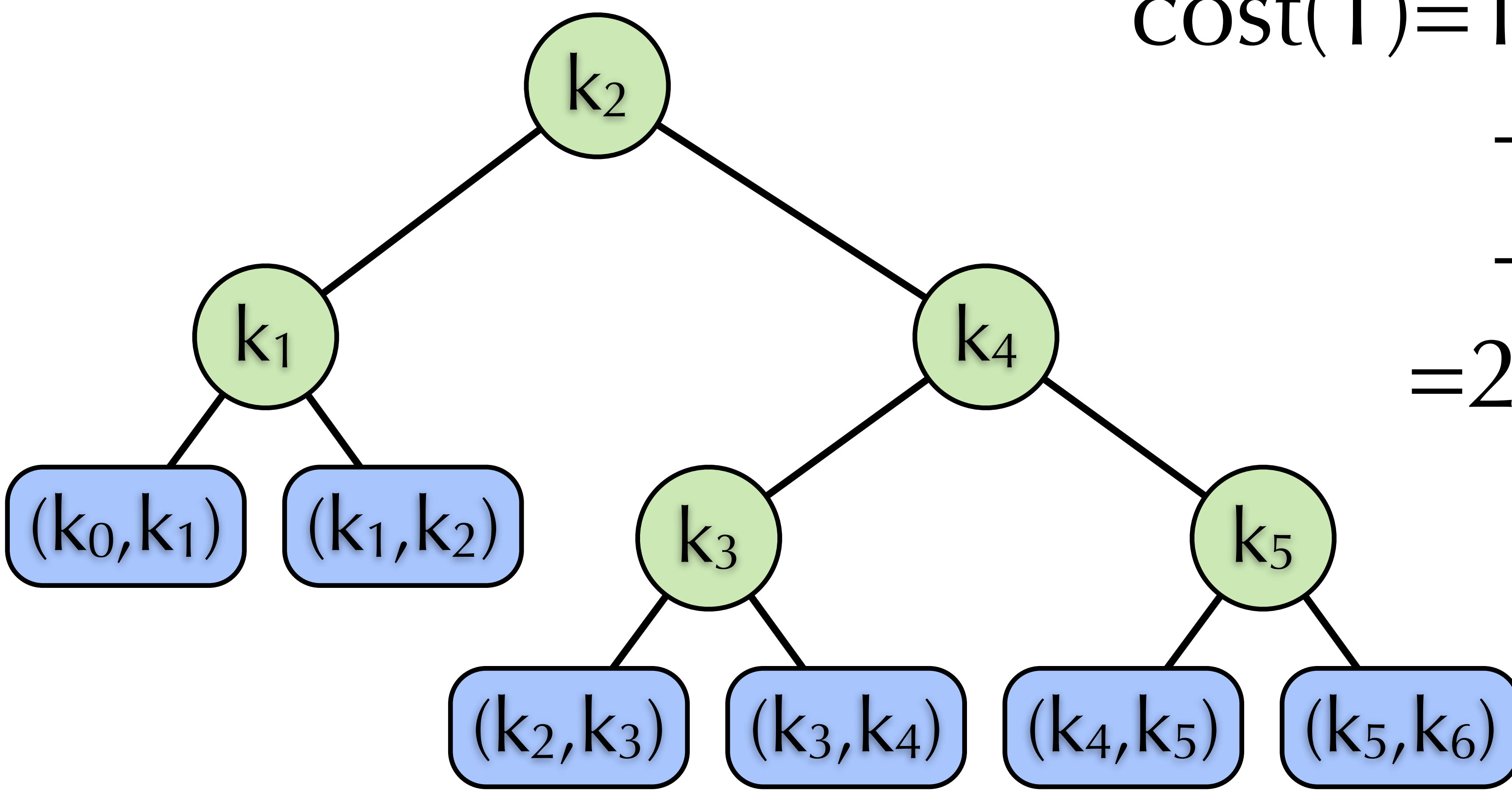
- Suppose a binary search tree has n keys $k_1<k_2<\ldots<k_n$.
- During the execution:
  - k_i is queried f_i times
  - (k_i,k_{i+1}) is queried g_i times
    - $k_0=-\infty$
    - $k_{n+1}=\infty$

Example: Optimal BST

- The cost of querying k_i: c_i
  - c_i is 1 plus the depth of node containing key k_i.
- The cost of querying (k_i,k_{i+1}): c'_i
  - c'_i is 1 plus the depth of node containing (k_i,k_{i+1}).
- The total cost: $cost(T)=\Sigma_{1\le i\le n}f_ic_i+\Sigma_{0\le i\le n}g_ic'_i$
- Goal: Minimizing the total cost.

Example: Optimal BST

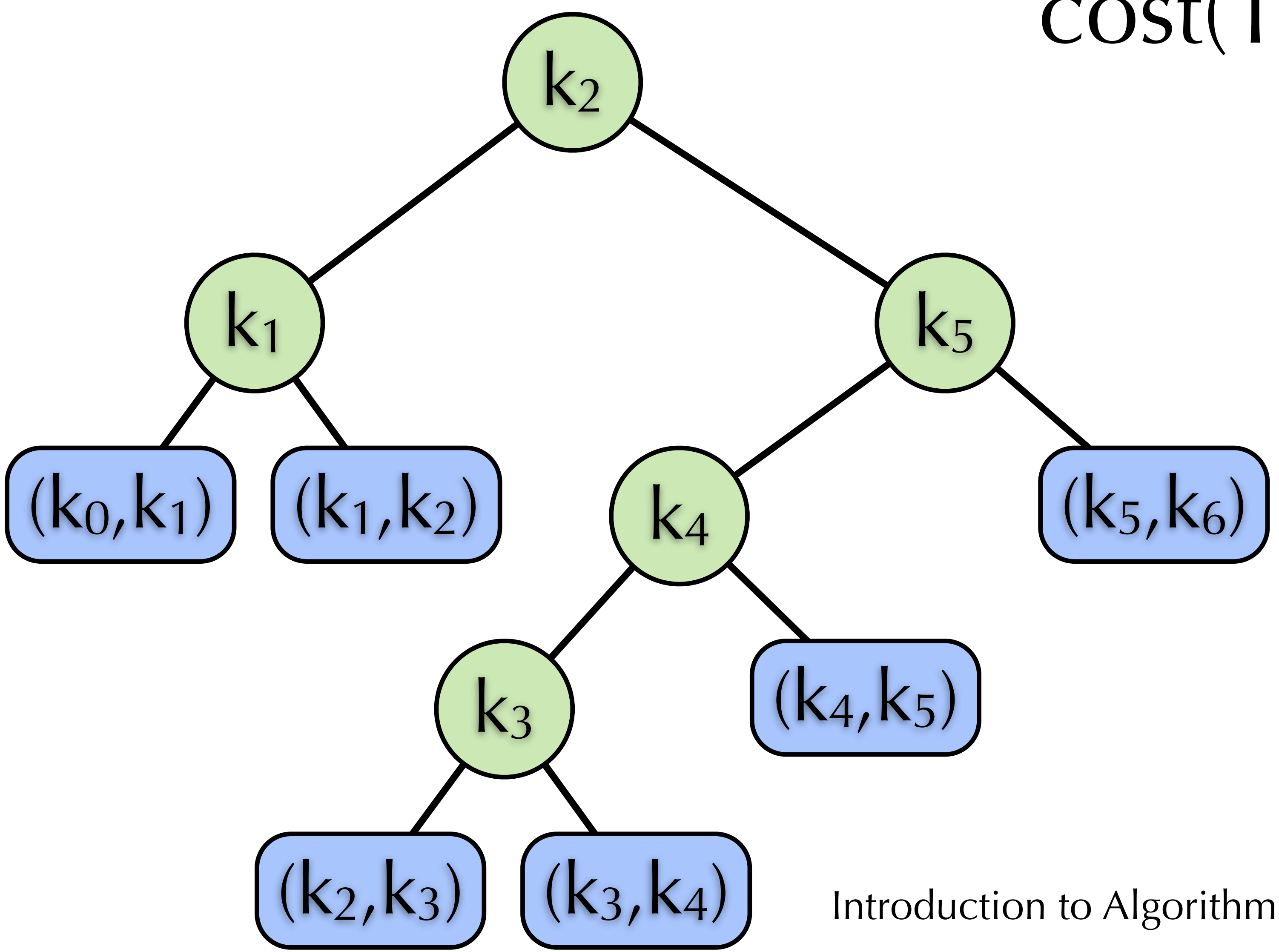
i	0	1	2	3	4	5
f_i		15	10	5	10	20
g_i	5	10	5	5	5	10



$$
\begin{aligned}
\text{cost}(T) = {} & 1\times f_2 + 2\times(f_1+f_4) \\
& +3\times(f_3+f_5+g_0+g_1) \\
& +4\times(g_2+g_3+g_4+g_5) \\
= {} & 280
\end{aligned}
$$

Example: Optimal BST

i	0	1	2	3	4	5
f_i		15	10	5	10	20
g_i	5	10	5	5	5	10



$$\begin{aligned} cost(T) = & 1\times f_2 + 2\times(f_1+f_5) \\ & +3\times(f_4+g_0+g_1) \\ & +4\times(f_3+g_5)+5(g_3+g_4) \\ = & 275 \end{aligned}$$

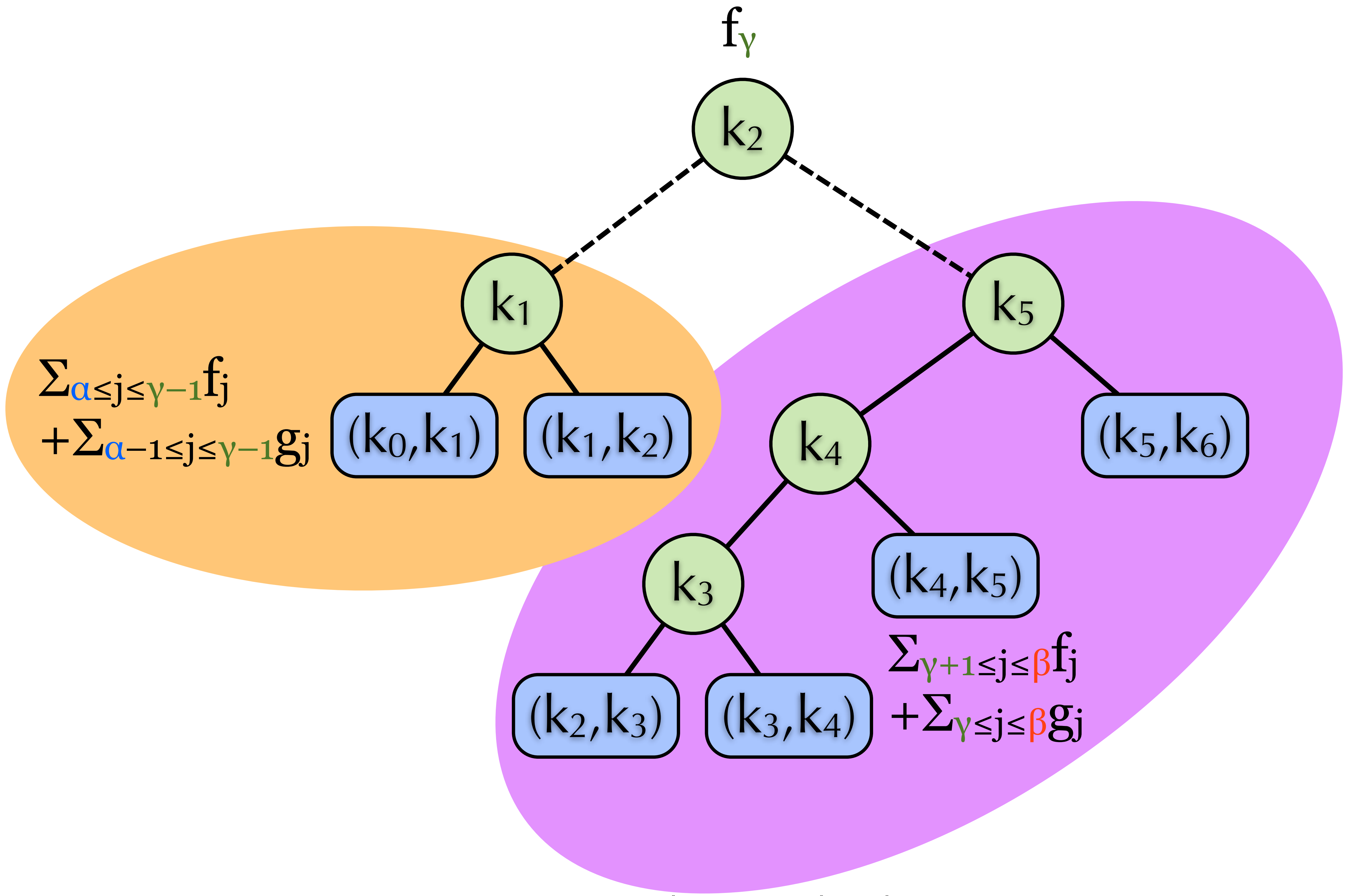
DP: cost(T)

- Termination: If $n=0$, return g_0.
- Divide: $k_1,\ldots,k_{i-1}$ & $k_{i+1},\ldots,k_n$ for $i \in \{1,\ldots,n\}$
- Conquer: Compute the answers
  - Let $p(i)$ be the answer of $k_1,\ldots,k_{i-1}$
  - Let $q(i)$ be the answer of $k_{i+1},\ldots,k_n$
- Combine: return $\Sigma_{1\le j\le n} f_j + \Sigma_{0\le j\le n} g_j + \min_{1\le i\le n}(p(i)+q(i))$

Dynamic Programming

- **Input**: $\langle f_\alpha,\ldots,f_\beta,g_{\alpha-1},\ldots,g_\beta\rangle$
- **Termination** $\langle g_{\alpha-1}\rangle$: return $g_{\alpha-1}$. $C(\alpha,\alpha-1)$
- **Divide**: Two types of subproblems
  - $L_\gamma=\langle f_\alpha,\ldots,f_\gamma,g_{\alpha-1},\ldots,g_\gamma\rangle$, for $\alpha-1\leq\gamma<\beta$
  - $R_\gamma=\langle f_\gamma,\ldots,f_\beta,g_{\gamma-1},\ldots,g_\beta\rangle$, for $\alpha<\gamma\leq\beta+1$
- **Conquer**: $C(\alpha,\gamma)=\mathrm{cost}(L_\gamma)$ & $C(\gamma,\beta)=\mathrm{cost}(R_\gamma)$
- **Combine**:
  $\Sigma_{\alpha\leq j\leq\beta}f_j+\Sigma_{\alpha-1\leq j\leq\beta}g_j+\min_{\alpha\leq\gamma\leq\beta}(C(\alpha,\gamma-1)+C(\gamma+1,\beta))$
- **Note**: $\mathrm{opt}(T)=C(1,n)$

$$\Sigma_{\alpha \le j \le \beta} f_j + \Sigma_{\alpha-1 \le j \le \beta} g_j$$



Dynamic Programming

$\Sigma_{\alpha\leq j\leq\beta} f_j + \Sigma_{\alpha-1\leq j\leq\beta} g_j$

$\alpha \backslash \beta$	0	1	2	3	4	5
1						
2						
3						
4						
5						
6						

$C(\alpha,\beta)$

$\alpha \backslash \beta$	0	1	2	3	4	5
1						
2						
3						
4						
5						
6						

Dynamic Programming

$\Sigma_{\alpha\leq j\leq\beta} f_j + \Sigma_{\alpha-1\leq j\leq\beta} g_j$

$\alpha \backslash \beta$	0	1	2	3	4	5
1	5	30	45	55	70	100
2		10	25	35	50	80
3			5	15	30	60
4				5	20	50
5					5	35
6						10

$C(\alpha,\beta)$

$\alpha \backslash \beta$	0	1	2	3	4	5
1						
2						
3						
4						
5						
6						

Dynamic Programming

$\Sigma_{\alpha\le j\le\beta} f_j + \Sigma_{\alpha-1\le j\le\beta} g_j$

$\alpha \backslash \beta$	0	1	2	3	4	5
1	5	30	45	55	70	100
2		10	25	35	50	80
3			5	15	30	60
4				5	20	50
5					5	35
6						10

$C(\alpha,\beta)$

$\alpha \backslash \beta$	0	1	2	3	4	5
1	5	45	90	125	175	275
2		10	40	70	120	200
3			5	25	60	130
4				5	30	90
5					5	50
6						10

Dynamic Programming

$d(\alpha,\beta)$: $\mathrm{minarg}_{\alpha\le\gamma\le\beta}(C(\alpha,\gamma-1)+C(\gamma+1,\beta))$

$\alpha \backslash \beta$	1	2	3	4	5
1	1	1	2	2	2
2		2	2	2	4
3			3	4	5
4				4	5
5					5

$C(\alpha,\beta)$

$\alpha \backslash \beta$	0	1	2	3	4	5
1	5	45	90	125	175	275
2		10	40	70	120	200
3			5	25	60	130
4				5	30	90
5					5	50
6						10

Time Complexity

- The original analysis:
  - Subproblems: n^2
  - Solve $\leq 2n$ subproblems to compute the optimal solution
  - Total: $O(n^3)$
- Knuth proved: $d(\alpha,\beta-1) \leq d(\alpha,\beta) \leq d(\alpha+1,\beta)$
  - Bonus: paper presentation

Time Complexity

- Knuth proved: $d(\alpha,\beta-1)\leq d(\alpha,\beta)\leq d(\alpha+1,\beta)$
- We only need to solve $O(1)$ subproblems on average to compute $C(\alpha,\beta)$!
  - Total time: $O(n^2)$
- Why?
  - $\min_{\alpha\leq\gamma\leq\beta}(C(\alpha,\gamma-1)+C(\gamma+1,\beta))$
    $=\min_{d(\alpha,\beta-1)\leq\gamma\leq d(\alpha+1,\beta)}(C(\alpha,\gamma-1)+C(\gamma+1,\beta))$
  - $1\leq d(\alpha,\beta)\leq n$
  - $d(1,1)\leq d(1,2)\leq d(2,2)\leq\ldots\leq d(n-1,n)\leq d(n,n)$
  - $d(1,2)\leq d(1,3)\leq d(2,3)\leq\ldots\leq d(n-1,n)$

Dynamic Programming

$d(\alpha,\beta)$: $\mathrm{minarg}_{\alpha\le\gamma\le\beta}(C(\alpha,\gamma-1)+C(\gamma+1,\beta))$

$\alpha \backslash \beta$	1	2	3	4	5
1	1	1	2	2	2
2		2	2	2	4
3			3	4	5
4				4	5
5					5

$1 \le d(1,4) \le d(1,5) \le d(2,5) \le 5$

$1 \le d(1,3) \le d(1,4) \le d(2,4) \le \ldots \le d(3,5) \le 5$

$1 \le d(1,2) \le d(1,3) \le d(2,3) \le \ldots \le d(4,5) \le 5$

$1 \le d(1,1) \le d(1,2) \le d(2,2) \le \ldots \le d(5,5) \le 5$

Dynamic Programming

$d(\alpha,\beta)$: $\text{minarg}_{\alpha\leq\gamma\leq\beta}(C(\alpha,\gamma-1)+C(\gamma+1,\beta))$

$\alpha \backslash \beta$	1	2	3	4	5
1	1	1	2	2	2
2		2	2	2	4
3			3	4	5
4				4	5
5					5

Note: the worst case is still $\Theta(n)$!

Total: $O(n^2)$ comparisons

Average: $O(n^2)/\Theta(n^2)=O(1)$

$O(n)$ comparisons

$O(n)$ comparisons

$O(n)$ comparisons

$O(n)$ comparisons

Amortized Analysis

- Methods
  - Aggregate analysis (17.1)
  - Accounting method (17.2)
  - Potential method (17.3)
- Examples
  - Stack operation: multi-pop
  - Incrementing a binary counter
  - Dynamic tables

Multi-Pop

- Stack supports multi-pop
  - empty: check if the stack is empty $\Theta(1)$
  - push(x): insert x $\Theta(1)$
  - pop(): extract an element $\Theta(1)$
  - multipop(k): extract k elements $O(k)$
- Suppose a stack is initially empty and $T(n)$ is the time complexity to execute n operations on it. $T(n)=O(n^2)$?
- $T(n)=\Theta(n)$, i.e., $\Theta(1)$ per operation.

Binary Counter

- A K-bit counter c can store $0\sim2^K-1$.
- Increment: c=c+1;
  - Cost: the number of bits changed
    - $O(\log m)$ if c store m
  - Example: increment $7=111_2$ cost 4.

  `00000111 → 00001000`
- Initially, c stores 0.
- C(n): the total cost to increment c to n.
- $C(n)=\Sigma_{1\leq m\leq n}O(\log m)=O(n\log n)$? $\Theta(n)$

Dynamic Table

- Table supports
  - Insert: insert an element
  - Delete: delete an element
- Implementation: array
- Load factor α: num/size num=#element
- Dynamic expansion: if the load factor is 1, then double the space. O(num)
- Insert without expansion: O(1)

Dynamic Table

x_1	

x_1	x_2

Insert x_2: 1 write

x_1	x_2		

Insert x_3: 1 allocation, 1 deallocation, 3 writes

x_1	x_2	x_3	

x_1	x_2	x_3	x_4

Insert x_4: 1 write

x_1	x_2	x_3	x_4				

Insert x_5: 1 allocation, 1 deallocation, 5 writes

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8

Insert x_9: 1 allocation, 1 deallocation, 9 writes

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9							

Dynamic Table

- Dynamic contraction: if the load factor is $\leq 1/4$, then halve the space. $O(n)$
- Delete without contraction: $O(1)$
- Time complexity of N consecutive operations to an empty table: $T(N)$
- Goal: $T(N)=\Theta(N)$
  - $\Theta(1)$ per operation

Aggregate Method

- Multi-pop stack S
  - Each element popped was pushed.
  - The total cost of multipop operations is no more than $\Theta(p)$ where p is the total number of push operations.
- $T(n)=\Omega(n)$ since there are n operations.
- $T(n)=p\Theta(1)+\Theta(p)\leq n\Theta(1)+\Theta(n)=\Theta(n)$
- $T(n)=\Theta(n)$

Aggregate Method

- Incrementing a binary counter
- When c stores $x2^{k+1}+2^k-1$ (a number ends in 01^k), then increment c will cost $\Theta(k)$d.
  - Ex:

    `01011001` → `01011010`

    `00000111` → `00001000`

    `01110011` → `01110100`

    `10011111` → `10100000`

Aggregate Method

- How many numbers in $\{0,\ldots,n-1\}$ are in the form $x2^{k+1}+2^k-1$? $\lfloor n/2^k \rfloor$
- $T(n) \leq \Sigma_{1\leq k\leq \log_2 n} k\lfloor n/2^k \rfloor < 2n$ ... goal
- How?
- $\Sigma_{1\leq k\leq s} k\lfloor n/2^k \rfloor \leq \Sigma_{1\leq k\leq s} kn/2^k$
- If $X=\Sigma_{1\leq k\leq s} kn/2^k$, then $X/2=\Sigma_{1\leq k\leq s} kn/2^{k+1}$
- $X-X/2=n/2+n/4+n/8+\ldots+n/2^s-n/2^{s+1}$ $<n/2+n/4+n/8+\ldots=n$
- $X/2<n$, so $T(n)\leq X<2n$.

Aggregate Method

- Dynamic table without contraction
- Insertion:
  - Without expansion: $\Theta(1)$
  - With expansion: $\Theta(s)$, where s is the size of the table after insertion.
- Observation: only at the first time we have $s=1+2^k$, we expand the table.
- In n operations, we only have at most $1+\log_2 n$ expansions.

Aggregate Method

- Suppose we have K expansions during these n operations.
- $T(n) \le n\Theta(1) + \Theta(2 + 2^2 + \ldots + 2^K)$
  $= \Theta(n) + \Theta(2^K)$
  $= \Theta(n) + O(2n)$ … $K \le 1 + \log_2 n$
  $= \Theta(n)$

Accounting Method

- Idea: assign different cost to each operation
  - This cost is called amortized cost c'.
  - Actual cost: c
  - Credit: c'−c
- During the execution, the total credit must be non-negative.
  - So the total amortized cost is an upper bound of the total actual cost.

Accounting Method

Operation	Actual cost	Amortized cost
empty	1	1
push	1	2
pop	1	0
multipop	min(s,k)	0

Note: you have to show that the total credit is always non-negative!

Accounting Method

Operation	Actual cost	Amortized cost
Change 0 to 1	1	2
Change 1 to 0	1	0

Note: you have to show that the total credit is always non-negative!

Accounting Method

Operation	Actual cost	Amortized cost
Insert	1	3
Delete	1	0
Expansion	s	0

Note: you have to show that the total credit is always non-negative!

Potential Method

- It might be hard to prove the total credit is always non-negative.
- Idea: your deposit account is always non-negative.
- Potential function Φ: deposit account
- Amortized cost: income
- Actual cost: expense
- Total credit is negative: bankrupt

Potential Function

- Potential function Φ (deposit)
  - Map a intermediate state into a value
  - $\Phi(S_0)=D_0$ (Initially, your deposit is 0.)
  - Always $\geq D_0$. (Your deposit ≥ 0)
- Example:
  - The stack size
  - The number of 1's
  - The number of elements

Potential Function

- Goal: to prove that the amortized costs are properly defined.
- n operations: $\sigma_1,\ldots,\sigma_n$.
  - σ_i transforms state S_{i-1} into S_i.
  - $c(\sigma_i)$: actual cost of σ_i
  - $c'(\sigma_i)$: amortized cost of σ_i
  - $c'(\sigma_i)=c(\sigma_i)+\Phi(S_i)-\Phi(S_{i-1})$.
- c' is properly defined: $\Phi(S_k)\geq\Phi(S_o)$

  $\Sigma_{1\leq i\leq k}c'(\sigma_i)=\Sigma_{1\leq i\leq k}(c(\sigma_i)+\Phi(S_i)-\Phi(S_{i-1}))$

  $=\Phi(S_k)-\Phi(S_o)+\Sigma_{1\leq i\leq k}c(\sigma_i)\geq\Sigma_{1\leq i\leq k}c(\sigma_i)$

Potential Function

- Goal: give a good amortized analysis
- Method: Find a good potential function Φ
- What is a good potential function?
  - IS a potential function.
  - The induced amortized costs are close.
- It might be hard to find a good potential function, but this method is one of the most powerful tool to analyze novel data structure.

Bonus

- Present: Show how to analyze the time complexity of splay trees.
- Present: Partial persistent data structures with O(1)-space and amortized O(1)-time overhead. http://courses.csail.mit.edu/6.851/spring12/lectures/L01.html

Example

- Multipop stack
  - $\Phi(S)$: the size of the stack
  - σ_i is empty: $c'(\sigma_i)=1+\Phi(S_i)-\Phi(S_{i-1})=1$
  - σ_i is push: $c'(\sigma_i)=1+\Phi(S_i)-\Phi(S_{i-1})=2$
  - σ_i is pop: $c'(\sigma_i)=1+\Phi(S_i)-\Phi(S_{i-1})=0$
  - σ_i is multipop: $c'(\sigma_i)=\min(s,k)+\Phi(S_i)-\Phi(S_{i-1})=0$

Example

- Incrementing a binary counter
  - $\Phi(S)$: the number of ones
  - σ_i increment $x2^{k+1}+2^k-1$ to $x2^{k+1}+2^k$:
    $c'(\sigma_i)=k+1+\Phi(S_i)-\Phi(S_{i-1})=k+1-(k-1)=2$
  - Example: k=5, actual cost=6
    `10011111` → `10100000`

Dynamic Table

- Support expansion & contraction
- s: size of allocated memory
- e: number of elements
- $\Phi(S_0)=0$ (Initially, $s=0$ and $e=0$)
- $\Phi(S_i)=2e_i-s_i$ if $2e_i \geq s_i$ (load factor $\alpha \geq 1/2$)
- $\Phi(S_i)=s_i/2-e_i$ if $2e_i \leq s_i$ (load factor $\alpha \leq 1/2$)
- Note: when $\alpha=1/2$, then $2e_i-s_i=0=s_i/2-e_i$.

Insertion w/o Expansion

$\Phi(S_i)=2e_i-s_i$ if $\alpha\geq 1/2$
$\Phi(S_i)=s_i/2-e_i$ if $\alpha\leq 1/2$

- $\alpha_{i-1}\geq 1/2$
  - $c'(\sigma_i)=1+\Phi(S_i)-\Phi(S_{i-1})$
    $=1+2e_i-2e_{i-1}-s_i+s_{i-1}$
    $=1+2e_i-2(e_i-1)-s_i+s_i=3$
- $\alpha_{i-1}<1/2$
  - $c'(\sigma_i)=1+\Phi(S_i)-\Phi(S_{i-1})$
    $=1+s_i/2-e_i-s_{i-1}/2+e_{i-1}$
    $=1+(e_i-1)-e_i+s_i/2-s_i/2$ … $e_{i-1}=e_i-1$
    $=0$

Insertion with Expansion

$\Phi(S_i) = 2e_i - s_i$ if $\alpha \geq 1/2$

- $\alpha_{i-1} = 1$
  - $c'(\sigma_i) = s_{i-1} + 1 + \Phi(S_i) - \Phi(S_{i-1})$
    $= s_{i-1} + 1 + 2e_i - 2e_{i-1} - s_i + s_{i-1}$
    $= s_{i-1} + 1 + 2e_i - 2(e_{i-1}) - 2s_{i-1} + s_{i-1}$
    $= 1 + 2(e_{i-1} + 1) - 2e_{i-1}$
    $= 3$

Deletion w/o Contraction

- $\alpha_i \geq 1/2$
  - $c'(\sigma_i) = 1 + \Phi(S_i) - \Phi(S_{i-1})$
    $= 1 + 2e_i - 2e_{i-1} - s_i + s_{i-1}$
    $= 1 + 2e_i - 2(e_i + 1) - s_i + s_i = -1$
- $\alpha_i < 1/2$
  - $c'(\sigma_i) = 1 + \Phi(S_i) - \Phi(S_{i-1})$
    $= 1 + s_i/2 - e_i - s_{i-1}/2 + e_{i-1}$
    $= 1 + (e_i + 1) - e_i + s_i/2 - s_i/2$ ... $e_{i-1} = e_i + 1$
    $= 2$

$\Phi(S_i) = 2e_i - s_i$ if $\alpha \geq 1/2$
$\Phi(S_i) = s_i/2 - e_i$ if $\alpha \leq 1/2$

Deletion with Contraction

$\boxed{\Phi(S_i)=s_i/2-e_i \text{ if } \alpha\leq 1/2}$

- Before deletion: $e_{i-1}-1=s_{i-1}/4$
- After contraction: $\alpha_i=1/2$
  - $c'(\sigma_i)=e_i+\Phi(S_i)-\Phi(S_{i-1})$
    $=e_i+s_i/2-e_i-s_{i-1}/2+e_{i-1}$
    $=s_i/2-s_{i-1}/2+e_i+1$ … $e_{i-1}=e_i+1$
    $=-s_i/2+e_i+1$ … $s_{i-1}=2s_i$
    $=-e_i+e_i+1$ … $s_i=2e_i$
    $=1$