Introduction to Data Management

Database Tuning

Alyssa Pittman
Based on slides by Jonathan Leang, Dan Suciu, et al

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle
Midterm results

- Scores released this morning via Gradescope
- Solutions on the website
- Regrade requests open until November 20
  - Please be specific/descriptive when asking for a question to be regraded
Midterm results

Midterm 100.0 points

<table>
<thead>
<tr>
<th>Minimum</th>
<th>Median</th>
<th>Maximum</th>
<th>Mean</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>41%</td>
<td>77.25%</td>
<td>100%</td>
<td>76.17%</td>
<td>13.12%</td>
</tr>
</tbody>
</table>
Midterm results

Most missed question: lj at 43% average

j) With respect to functional dependencies, a relation can have multiple keys but only one superkey.

True  False
Best question: 3a at 97% average (closely followed by 3c, you’re RA rockstars!)

a) (13 points) The restaurant is trying to have more allergy-friendly dishes for their customers at all meals. They wrote the following query to find the number of allergens in their dishes in various categories:

```sql
SELECT d.name, d.category, COUNT(*) AS cnt
FROM Ingredient i, Dish d, IngredientIn ii
WHERE i.iid = ii.iid AND d.did = ii.did
    AND i.allergen = 1
GROUP BY d.did, d.name, d.category
HAVING COUNT(*) > 1;
```

Write a Relational Algebra expression in the form of a logical query plan (you may draw a tree) that is equivalent to the SQL query.
Goals for Today

▪ We learned about index structures and selectivity...
▪ ...now we’ll practice cost estimation using index-based joins.
Recap: Indexing

An **index** is an additional file allowing **fast access** to records given a **search key**.

It stores **(key, value) pairs:**

**(attribute, pointer to the record)**

<table>
<thead>
<tr>
<th>Key</th>
<th>Value1</th>
<th>Value2</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>John</td>
<td>Doe</td>
</tr>
<tr>
<td>20</td>
<td>Jane</td>
<td>Doe</td>
</tr>
<tr>
<td>50</td>
<td>Alice</td>
<td>Foo</td>
</tr>
<tr>
<td>200</td>
<td>Bob</td>
<td>Bar</td>
</tr>
</tbody>
</table>
Recap: Clustered vs Unclustered Index

A **clustered index** is one that has the same key ordering as what is on disk (one per table).
Recap: Clustered vs Unclustered Index

- An **unclustered index** may exist without any ordering on disk (any number per table)

**Sequential File with a different key** or **Heap File**
Recap – Making Cost Estimations

- RDBMS keeps statistics about our tables
  - \( B(R) = \# \text{ of blocks} \) in relation \( R \)
  - \( T(R) = \# \text{ of tuples} \) in relation \( R \)
  - \( V(\text{attr}, R) = \# \text{ of distinct values} \) of attr in \( R \)
Recap: Selectivity factor

- **Selectivity Factor** \((X)\) → Proportion of total data needed

- Assuming uniform distribution of data values on numeric attribute \(a\) in table \(R\), if the condition is:
  - \(a=c\) \(\rightarrow\) \(X \approx \frac{1}{V(a,R)}\)
  - \(a<c\) \(\rightarrow\) \(X \approx \frac{c-\min(a,R)}{\max(a,R)-\min(a,R)}\)
  - \(c_1<a<c_2\) \(\rightarrow\) \(X \approx \frac{c_2-c_1}{\max(a,R)-\min(a,R)}\)
  - \(\text{cond1 AND cond2}\) \(\rightarrow\) \(X \approx X_1 \times X_2\)

- Disclaimer: More thorough selectivity estimation will use a histogram
Outline

- Index-based scans
- Database tuning
- Index join cost estimation
- Multiple joins cost estimation
Index-Based Selection

- For reference, a full sequential scan of data costs $B(R)$ IOs.

- Provided some condition to read data:
  - Full **sequential scan** $\subseteq B(R)$
  - Scan on **clustered index** $\subseteq X*B(R)$
    - Able to read a contiguous chunk of the file
  - Scan on **unclustered index** $\subsetneq X*T(R)$
    - Worst case would read a different block everytime
Sequential Scan

Assume a block holds 4 tuples. I want tuples associated with values 40–85. Without an index, finding a value must be done the “old fashioned way”
Sequential Scan

Assume a block holds 4 tuples. I want tuples associated with values 40–85. Without an index, finding a value must be done the “old fashioned way.”

Disk

| d1 | d2 | d3 | d4 | d5 | d6 | d7 | d8 | d9 | d10 | d11 | d12 | d13 |
Sequential Scan

Assume a block holds 4 tuples. I want tuples associated with values 40-85. Without an index, finding a value must be done the “old fashioned way”
Sequential Scan

Assume a block holds 4 tuples. I want tuples associated with values 40–85. Without an index, finding a value must be done the “old fashioned way”
Sequential Scan

Assume a block holds 4 tuples. I want tuples associated with values 40–85. Without an index, finding a value must be done the “old fashioned way.”
Sequential Scan

Assume a block holds 4 tuples. I want tuples associated with values 40–85. Without an index, finding a value must be done the “old fashioned way”

Total cost: $B(R)$

Disk
Assume a block holds 4 tuples. I want tuples associated with values 40–85. With a clustered index, I start scanning blocks in the range they are at.
Assume a block holds 4 tuples. I want tuples associated with values 40-85. With a clustered index, I start scanning blocks in the range they are at.
Indexed

Assume a block holds 4 tuples. I want tuples associated with values 40-85. With a clustered index, I start scanning blocks in the range they are at

Total cost: $X \times B(R)$

Sequential File

Index

Clustered Index Scan

November 4, 2019
Assume a block holds 4 tuples. I want tuples associated with values 40-85. With an unclustered index, I scan tuples wherever they occur.
Assume a block holds 4 tuples. I want tuples associated with values 40–85. With an unclustered index, I scan tuples wherever they occur.
Unclustered Index Scan

Assume a block holds 4 tuples. I want tuples associated with values 40–85. With an unclustered index, I scan tuples wherever they occur.

Sequential File with a different key or Heap File
Unclustered Index Scan

Assume a block holds 4 tuples. I want tuples associated with values 40-85. With an unclustered index, I scan tuples wherever they occur.

Sequential File with a different key or Heap File

Lucky cache hit!
Unclustered Index Scan

Assume a block holds 4 tuples. I want tuples associated with values 40–85. With an unclustered index, I scan tuples wherever they occur.

Sequential File with a different key or Heap File
Unclustered Index Scan

Assume a block holds 4 tuples. I want tuples associated with values 40-85. With an unclustered index, I scan tuples wherever they occur

Total cost: $X \times T(R)$

Sequential File with a different key or Heap File
Index Expectations

- Using an index in the wrong scenario can lead to a slowdown!

- Common example:

  Full sequential scan vs unclustered index scan with **high X value** and/or **small tuple size** (large T(R):B(R) ratio)

  Known:
  - B(R) = 100
  - T(R) = 10000

  Consider a query with X=1/10

  **Sequential scan**
  - = B(R)
  - = **100**

  **Unclustered index scan**
  - = X*T(R)
  - = 1/10 * 10000
  - = **1000**
Index Expectations

- Using an index in the wrong scenario can lead to a slowdown!

- Common example:

  Full sequential scan vs unclustered index scan with **high X value** and/or **small tuple size** (large $T(R):B(R)$ ratio)

Having indexes doesn’t mean you will see a speedup!

Known:
- $B(R) = 100$
- $T(R) = 10000$

Consider a query with $X=1/10$

<table>
<thead>
<tr>
<th>Sequential scan</th>
<th>Unclustered index scan</th>
</tr>
</thead>
<tbody>
<tr>
<td>$= B(R)$</td>
<td>$= X*T(R)$</td>
</tr>
<tr>
<td>$= 100$</td>
<td>$= 1/10 * 10000$</td>
</tr>
<tr>
<td>$= 1000$</td>
<td>$= 1000$</td>
</tr>
</tbody>
</table>
**Index Expectations**

- **Sequential disk reads are faster than random ones**
  - Cost \(\sim 1-2\%\) random scan = full sequential scan

![Graph showing the cost to scan different types of data](image-url)
CREATE TABLE Users (  
id INT,  
age INT,  
score INT);

CREATE INDEX U_age ON Users(age)

CREATE INDEX U_age_score ON Users(age, score)

CREATE CLUSTERED INDEX U_score_age ON Users(score, age)
CREATE TABLE Users (  
id INT,  
age INT,  
score INT);

CREATE INDEX U_age ON Users(age)

CREATE INDEX U_age_score ON Users(age, score)

CREATE CLUSTERED INDEX U_score_age ON Users(score, age)
CREATE TABLE Users (  id INT,  age INT,  score INT);

CREATE INDEX U_age ON Users(age)

CREATE INDEX U_age_score ON Users(age, score)

CREATE CLUSTERED INDEX U_score_age ON Users(score, age)
Create Indexes in SQL

CREATE TABLE Users (  
id INT,  
age INT,  
score INT);

CREATE INDEX U_age ON Users(age);

CREATE INDEX U_age_score ON Users(age, score);

CREATE CLUSTERED INDEX U_score_age ON Users(score, age);

Does U_age_score work for these?

SELECT *  
FROM Users  
WHERE age = 21 and score > 90;

SELECT *  
FROM Users  
WHERE age = 21;

SELECT *  
FROM Users  
WHERE score > 90;
CREATE TABLE Users (
    id INT,
    age INT,
    score INT);

CREATE INDEX U_age ON Users(age)

CREATE INDEX U_age_score ON Users(age, score)

CREATE CLUSTERED INDEX U_score_age ON Users(score, age)

Does U_age_score work for these?

SELECT *
FROM Users
WHERE age = 21 and score > 90;

SELECT *
FROM Users
WHERE age = 21;

SELECT *
FROM Users
WHERE score > 90;
CREATE TABLE Users (  id INT,  age INT,  score INT);

CREATE INDEX U_age ON Users(age)

CREATE INDEX U_age_score ON Users(age, score)

CREATE CLUSTERED INDEX U_score_age ON Users(score, age)
CREATE TABLE Users (  
id INT,  
age INT,  
score INT);

CREATE INDEX U_age ON Users(age)

CREATE INDEX U_age_score ON Users(age, score)

CREATE CLUSTERED INDEX U_score_age ON Users(score, age)

Does U_score_age work for these?

SELECT *  
FROM Users  
WHERE age = 21 and score > 90;

SELECT *  
FROM Users  
WHERE age = 21;

SELECT *  
FROM Users  
WHERE score > 90;
CREATE TABLE Users (  
id INT,  
age INT,  
score INT);

CREATE INDEX U_age ON Users(age)

CREATE INDEX U_age_score ON Users(age, score)

CREATE CLUSTERED INDEX U_score_age ON Users(score, age)

Fine to create indexes on non-numeric data (just not discussed in this class)

Unclustered by default

Order specifies precedence in sorting

Reorders data on disk! (Fails if another clustered index exists)
Leveraging Indexes

- Often for applications, workloads can be well described
  - Canvas Gradebook
    - View grades query for grades by student id
  - Data visualization software (e.g. Tableau)
    - 2D plot query on graph axis bounds

- Create indexes to match expected query workload
Leveraging Indexes

Make attribute K a search key if you have queries where the WHERE clause contains:

- An exact match on K
- A range predicate on K
- A join on K
CREATE TABLE Users (  
id INT PRIMARY KEY,  
age INT,  
score INT, ...);  

What indexes could we make on Users?

Expecting 1000 exec/day
SELECT *  
FROM Users, Assets  
WHERE Users.id = Assets.uid

Expecting 1000 exec/day
SELECT *  
FROM Users  
WHERE Users.score > 95

Expecting 10 exec/day
SELECT *  
FROM Users  
WHERE Users.age > 21
CREATE TABLE Users (  
id INT PRIMARY KEY,  
age INT,  
score INT, ...);

What indexes could we make on Users?

IDs are unique so an unclustered index would do fine.

Expecting 1000 exec/day
SELECT *  
FROM Users, Assets  
WHERE Users.id = Assets.uid

Expecting 1000 exec/day
SELECT *  
FROM Users  
WHERE Users.score > 95

Expecting 10 exec/day
SELECT *  
FROM Users  
WHERE Users.age > 21
Leveraging Indexes

CREATE TABLE Users (  
id INT PRIMARY KEY,  
age INT,  
score INT, ...);  

What indexes could we make on Users?

IDs are unique so an unclustered index would do fine.

Expecting 1000 exec/day
SELECT *  
FROM Users, Assets  
WHERE Users.id = Assets.uid

This range query would benefit from a clustered index on score

Expecting 1000 exec/day
SELECT *  
FROM Users  
WHERE Users.score > 95

Only one can exist!

Expecting 10 exec/day
SELECT *  
FROM Users  
WHERE Users.age > 21

This range query would benefit from a clustered index on age
CREATE TABLE Users (  id INT PRIMARY KEY,  age INT,  score INT, ...);

Expecting 1000 exec/day
SELECT *  
FROM Users, Assets  
WHERE Users.id = Assets.uid

Expecting 1000 exec/day
SELECT *  
FROM Users  
WHERE Users.score > 95

Expecting 10 exec/day
SELECT *  
FROM Users  
WHERE Users.age > 21

What indexes could we make on Users?

IDs are unique so an unclustered index would do fine.

Things to consider:
• How frequently are these queries executed?  
• Do either of these queries need to be returned ASAP?  
• What is the expected result size for each query?  
• How often is data inserted into this table?
CREATE TABLE Users (  id INT PRIMARY KEY,  age INT,  score INT, ...);

What indexes could we make on Users?

IDs are unique so an unclustered index would do fine.

Expecting 1000 exec/day
SELECT *  
FROM Users, Assets  
WHERE Users.id = Assets.uid

Expecting 1000 exec/day
SELECT *  
FROM Users  
WHERE Users.score > 95

Without more information, default to clustering on the index that will be used more (clustered index on score)

Expecting 10 exec/day
SELECT *  
FROM Users  
WHERE Users.age > 21
CREATE TABLE Users (  
id INT PRIMARY KEY,  
age INT,  
score INT, ...);

What indexes could we make on Users?

IDs are unique so an unclustered index would do fine.

Hack:
• Create a covering index primarily keyed on score
• Create a covering index primarily keyed on age

Expecting 1000 exec/day
SELECT *  
FROM Users, Assets  
WHERE Users.id = Assets.uid

Expecting 1000 exec/day
SELECT *  
FROM Users  
WHERE Users.score > 95

Expecting 10 exec/day
SELECT *  
FROM Users  
WHERE Users.age > 21
Leveraging Indexes

CREATE TABLE Users (  
id INT PRIMARY KEY,  
age INT,  
score INT, ...);

What indexes could we make on Users?

Hack:
• Create a covering index primarily keyed on score
• Create a covering index primarily keyed on age

IDs are unique so an unclustered index would do fine.

Essentially a sorted copy of the table. Fast but space inefficient and table updates are slow.

Expecting 1000 exec/day
SELECT *  
FROM Users, Assets  
WHERE Users.id = Assets.uid

Expecting 1000 exec/day
SELECT *  
FROM Users  
WHERE Users.score > 95

Expecting 10 exec/day
SELECT *  
FROM Users  
WHERE Users.age > 21
Choosing how to configure a database system is an interesting (i.e. hard) problem.

A database that is used by many people will often need one or more dedicated personnel to manage it (Database Administrator):

- Logical design (multi-team coordination)
- Physical design (hardware and system considerations)
- Permission management (visibility and security)
- Integration (company acquisitions and mergers)
- ...
Index-Based Equijoin

- Assume index exists on the join attribute a of S

```
SELECT *  
FROM R, S  
WHERE R.a = S.a
```

- **Clustered Index Join**
  - Perform a clustered index scan for each tuple of R
  - \( B(R) + T(R)(X*B(S)) = B(R) + T(R)(B(S)/V(a,S)) \)

- **Unclustered Index Join**
  - Perform an unclustered index scan for each tuple of R
  - \( B(R) + T(R)(X*T(S)) = B(R) + T(R)(T(S)/V(a,S)) \)
Index-Based Equijoin

- Assume index exists on the join attribute a of S

```
SELECT *  
FROM R, S  
WHERE R.a = S.a
```

- **Clustered Index Join**
  - Perform a clustered index scan for each tuple of R
  - \( B(R) + T(R)(X*B(S)) = B(R) + T(R)(B(S)/V(a,S)) \)

- **Unclustered Index Join**
  - Perform an unclustered index scan for each tuple of R
  - \( B(R) + T(R)(X*T(S)) = B(R) + T(R)(T(S)/V(a,S)) \)

Can't scan per block of R since tuples in blocks don't have the same attribute values
Multiple Joins

▪ **Pipelined Execution**
  - Tuples are processed through the entire query plan
  - Fast

▪ **Blocking Execution**
  - Subplans are computed and stored before parent operation can start
  - Simple
Pipelined Execution

- Iterator interface of RA operators (Volcano Iterator Model)
  - `open()` on every operator at start
  - `next()` to get the next tuple from a child operator or input table
  - `close()` on every operator at end
Pipelined Execution Example

- Iterator interface of RA operators (Volcano Iterator Model)
Pipelined Execution Example

- Iterator interface of RA operators (Volcano Iterator Model)

```
\sigma_{R.b=5} \quad \text{open()}
\bowtie S.a=T.a
\bowtie R.a=S.a
R \quad S \quad T
```
Pipelined Execution Example

- Iterator interface of RA operators (Volcano Iterator Model)
Pipelined Execution Example

- Iterator interface of RA operators (Volcano Iterator Model)
Pipelined Execution Example

- Iterator interface of RA operators (Volcano Iterator Model)
Pipelined Execution Example

- Iterator interface of RA operators (**Volcano Iterator Model**)
Pipelined Execution Example

- Iterator interface of RA operators (Volcano Iterator Model)

```
R.a=S.a
\sigma_{R.b=5}  next()
\bowtie_{S.a=T.a}  next()
```

```
R
S
T
```
• Iterator interface of RA operators (Volcano Iterator Model)

next() implementation will depend on algorithm used (CSE 444)
**Pipelined Execution Example**

- Iterator interface of RA operators (Volcano Iterator Model)
Pipelined Execution Example

- Iterator interface of RA operators *(Volcano Iterator Model)*
Pipelined Execution Example

- Iterator interface of RA operators (Volcano Iterator Model)
### Pipelined Execution Example

- **Iterator interface of RA operators** *(Volcano Iterator Model)*

```
R: □ R.a = S.a
  □ next()
  next()...

S: □ S.a = T.a
  □ next()
  next()...

T: □ next()
  next()...

σ_{R.b=5}

next()
```
Pipelined Execution Example

- Iterator interface of RA operators (Volcano Iterator Model)
Pipelined Execution Example

- Iterator interface of RA operators (**Volcano Iterator Model**)
Estimated IO cost for a plan can be lowered under pipelined execution.

Generated tuples are in main memory so future access is “free”.

\[
\begin{align*}
B(R) + B(R)B(S) & \quad \leftarrow \leftarrow (\text{block-at-a-time nested loop join})
\end{align*}
\]

\[
\begin{align*}
B(RS) + B(RS)B(T) & \quad \leftarrow \leftarrow (\text{block-at-a-time nested loop join})
\end{align*}
\]

\[
\begin{align*}
B(RST) & \quad \sigma_{R.b=5}
\end{align*}
\]

November 4, 2019

Indexes
- Estimated IO cost for a plan can be lowered under pipelined execution
- Generated tuples are in main memory so future access is “free”

\[
B(R) + B(R)B(S) \rightarrow B(RS) + B(RS)B(T) \rightarrow B(RS + T) \rightarrow B(RST)
\]

\[
\sigma_{R.b=5}
\]

\[
\sigma_{S.a=T.a}
\]

\[
\sigma_{R.a=S.a}
\]

Everything but file scans are done “on the fly” (free)

Reading RS is free
Estimated IO cost for a plan can be lowered under pipelined execution.

Generated tuples are in main memory so future access is “free”.

\[ \text{Est. Cost} = B(R) + B(R)B(S) + B(RS)B(T) \]

Everything but file scans are done “on the fly” (free).

Reading RS is free.
Example Problem

Assuming:
- Indexes and tuples fit in memory
- Unclustered indexes on R.A and R.B
- Clustered index on S.B, unclustered index on S.C.
- Clustered index on T.C, unclustered index on T.D.

Cost of unclustered index scan:
\[
\begin{align*}
\text{Cost} &= X \times T(R) \\
&= \left(1/V(R,A)\right) \times T(R) \\
&= \left(1/5 \times 10^4\right) \times 10^5 \\
&= 2
\end{align*}
\]
Example Problem

Assuming:
- Indexes and tuples fit in memory
- Unclustered indexes on R.A and R.B
- Clustered index on S.B, unclustered index on S.C.
- Clustered index on T.C, unclustered index on T.D.

\[ T(T_1) = 2 \]

Cost of unclustered index scan:
\[ X \times T(R) = \left( \frac{1}{V(R,A)} \right) \times T(R) = \left( \frac{1}{5 \times 10^4} \right) \times 10^5 = 2 \]
**Example Problem**

Cost of clustered index join:
\[
\begin{align*}
T(\text{TI}) & = B(\text{TI}) + T(\text{TI}) \times X \times B(\text{S}) \\
& = B(\text{TI}) + T(\text{TI}) \times \left(\frac{1}{V(S,B)}\right) \times B(\text{S}) \\
& = 0 + 2 \times \left(\frac{1}{3 \times 10^3}\right) \times 3000 \\
& = 2
\end{align*}
\]

Assuming:
- Indexes and tuples fit in memory
- Unclustered indexes on R.A and R.B
- Clustered index on S.B, unclustered index on S.C.
- Clustered index on T.C, unclustered index on T.D.

Cost of unclustered index scan:
\[
\begin{align*}
X \times T(R) & = \left(\frac{1}{V(R,A)}\right) \times T(R) \\
& = \left(\frac{1}{5 \times 10^4}\right) \times 10^5 \\
& = 2
\end{align*}
\]
Example Problem

Assuming:
- Indexes and tuples fit in memory
- Unclustered indexes on R.A and R.B
- Clustered index on S.B, unclustered index on S.C.
- Clustered index on T.C, unclustered index on T.D.

\[
T(T2) = T(T1) \times T(S) \times X \\
= 2 \times 6 \times 10^6 / 3 \times 10^3 \\
= 4000
\]

Cost of unclustered index scan:
\[
= X \times T(R) \\
= (1/V(R,A)) \times T(R) \\
= (1/5 \times 10^4) \times 10^5 \\
= 2
\]

\[
T(T1) = 2
\]

Cost of clustered index join:
\[
= B(T1) + T(T1) \times X \times B(S) \\
= B(T1) + \\
T(T1) \times (1/V(S,B)) \times B(S) \\
= 0 + 2 \times (1/3 \times 10^3) \times 3000 \\
= 2
\]

\[
T(C,D) = 40000 \\
V(T,C) = 5 \times 10^4
\]
Example Problem

\[
T(T2) = T(T1) \times T(S) \times X \\
= 2 \times 6 \times 10^6 / 3 \times 10^3 \\
= 4000
\]

Cost of clustered index join:

\[
= B(T1) + T(T1) \times X \times B(S) \\
= B(T1) + T(T1) \times (1/V(S,B)) \times B(S) \\
= 0 + 2 \times (1/3 \times 10^3) \times 3000 \\
= 2
\]

\[
T(T1) = 2
\]

Cost of unclustered index scan:

\[
= X \times T(R) \\
= (1/V(R,A)) \times T(R) \\
= (1/5 \times 10^4) \times 10^5 \\
= 2
\]

Assuming:
- Indexes and tuples fit in memory
- Unclustered indexes on R.A and R.B
- Clustered index on S.B, unclustered index on S.C.
- Clustered index on T.C, unclustered index on T.D.

Cost of clustered index join:

\[
= B(T2) + T(T2) \times X \times B(T) \\
= B(T2) + T(T2) \times (1/V(T,C)) \times B(T) \\
= 0 + 4 \times 10^3 \times (1/2 \times 10^4) \times 4 \times 10^4 \\
T(C,D) = 8 \times 10^3 = 8000
\]
Example Problem

Assuming:
- Indexes and tuples fit in memory
- Indexes on R.A
- Index on S.B, index on S.C.
- Index on T.C, index on T.D.

Cost of clustered index join:
\[ = B(T2) + T(T2) \cdot X \cdot B(T) \]
\[ = B(T2) + T(T2) \cdot \left(\frac{1}{V(T,C)}\right) \cdot B(T) \]
\[ = 0 + 4 \cdot 10^3 \cdot \left(\frac{1}{2 \cdot 10^4}\right) \cdot 4 \cdot 10^4 \]
\[ = 8 \cdot 10^3 = 8000 \]

Total cost = 2 + 2 + 8000 = 8004