ADS2 — Week 7 Notes & Solutions (Hashing)

Field	Value
Title	ADS2 — Week 7: Hashing (Dictionaries, Chained Hashing, Linear Probing, Hash Functions)
Date	2025-11-07
Author	Mads Richardt (prepared with GPT-5 Thinking)
Sources used	weekplan7.pdf (pp. 1–2); kt.pdf (hashing chapters); slide images hashing-01…10.png
Week plan filename	weekplan7.pdf

General Methodology and Theory

- **Dictionaries & goal.** Maintain a dynamic set $S \subseteq U$ with operations SEARCH, INSERT, DELETE in $O(1)$ expected time using $O(n)$ space. Store optional satellite data per key.
- **Hashing idea.** Compute an address $h(x) \in \{0,\dots,m-1\}$ and keep buckets $A[0\dots m-1]$. Need h that spreads S "approximately evenly."
- **Chained hashing.** Each A[i] stores a (typically singly linked) list. Operations run in $O(1 + |A[h(x)]|)$. With load factor $\alpha = n/m$ and simple-uniform hashing, $E[|A[h(x)]|] = \alpha$, so expected $O(1)$.
- **Open addressing (linear probing).** Keep a single array of size m; collisions are resolved by scanning cyclically until an empty slot is found. Clustering matters; expected time $\approx O(1/(1-\alpha)^2)$ under simple models.
- **Simple uniform hashing.** For any fixed $x \neq y$, $\Pr[h(x)=h(y)] = 1/m$. Yields expected chain length $1 + (n-1)/m$ for a bucket containing x.
- **Universal hashing.** Choose h at random from a family H with pairwise-independence guarantee: $\forall x\neq y$, $\Pr_h[h(x)=h(y)] \leq 1/m$. Classic families:
- For prime $p > |U|$, $h_{a,b}(x) = ((a\cdot x + b) \bmod p) \bmod m$, where $a\in\{1,\dots,p-1\}$, $b\in\{0,\dots,p-1\}$.
- Dot-product hashing for large universes: represent x in base m as vector and use $h_a(x) = (a\cdot x) \bmod m$.
- **Deletion rules.**
- Chaining: remove node from its list.
- Linear probing: deletion must **reinsert** the following cluster or use a **DELETED** tombstone to preserve successful searches.

Core invariants. (i) Every stored key is reachable by SEARCH; (ii) No duplicates; (iii) α stays in a target band via resize (e.g., m doubles/halves at thresholds).

Pseudocode skeletons.

```
Algorithm: chained_insert
Input: table A[0..m-1], hash h(·), key x
```

```
Output: A with x inserted if absent

i ← h(x)
if x ∈ A[i]: return
prepend x to list A[i]
// Time: O(1 + |A[i]|); Space: O(1)
```

```
Algorithm: linear_probe_search
Input: array A[0..m-1] (⊥ for empty, ◊ for tombstone), hash h, key x
Output: index j with A[j]=x or ⊥ if absent

for k = 0..m−1:
  j ← (h(x)+k) mod m
  if A[j] = ⊥: return ⊥
  if A[j] = x: return j
return ⊥
// Time: O(cluster length)
```

Notes (slides-first; compact)

Topics match the 10 slide images (hashing-01...10.png): dictionaries; chained hashing (idea, ops, time, space); linear probing (ops, time/variants); hash functions; simple-uniform hashing (indicator-variable proof); universal hashing (lemmas, families, dot-product hashing; theorem: O(n) space, O(1) expected time/op). Use these images for visual walk-throughs and in-class examples.

Key takeaways. - With **chaining + simple-uniform hashing** and m = Θ(n), all three operations run in expected O(1). - **Linear probing** is cache-friendly but sensitive to clustering; keep α ≤ ~0.7 and resize. - **Universal hashing** provides collision bounds independent of S; pick h at operation start or per table resize.

Coverage Table (enumerated from weekplan7.pdf)

Due to poor text extraction (fonts without spaces), we apply the *scanned/low-text fallback*. The plan lists seven numbered tasks under “Exercises/Problems”. We enumerate them conservatively; if any label is off, replace the Title/Label with the exact line from the plan and we will re-align.

Weekplan ID	Canonical ID	Title/Label (verbatim if readable)	Assignment Source	Text Source	Status
1	—	Chained hashing: build table & basic ops	weekplan7.pdf p.1	hashing-03.png, hashing-04.png	Solved

Weekplan ID	Canonical ID	Title/Label (verbatim if readable)	Assignment Source	Text Source	Status
2	—	Chained hashing: time & space; expected bucket length	weekplan7.pdf p.1	hashing-05.png, hashing-09.png	Solved
3	—	Linear probing: insert/search trace	weekplan7.pdf p.1	hashing-06.png, hashing-08.png	Solved
4	—	Lazy deletion in linear probing	weekplan7.pdf p.2	hashing-08.png	Solved
5	—	Simple uniform hashing: indicator-variable proof	weekplan7.pdf p.2	hashing-09.png	Solved
6	—	Universal hashing: family & collision bound	weekplan7.pdf p.2	hashing-10.png; kt.pdf	Solved
7	—	**Puzzle:** "Billy & the carrots" (expected visits)	weekplan7.pdf p.2	—	BLOCKER

MISMATCH/Blockers. If the plan has additional or differently worded items, paste their exact lines (or a screenshot of the exercise block) and we will update the Coverage Table. For Item 7 we need the precise statement (parameters such as number of bushes, carrot placement model, with/without replacement) to finalize the expectation.

Solutions

Slides-first; textbook variants (KT) are noted where helpful.

Exercise 1 — Chained hashing: build table & basic ops

Assignment Source: weekplan7.pdf p.1
Text Source: hashing-03/04.png (insertion demo)

- **Setup.** Let m=10 and h(x)=x mod 10; insert S={1,16,41,54,66,96} (from slide). Place each in A[h(x)].
- **Trace.** A[1]: 1, 41 → 1→41; A[6]: 16, 66, 96 → 16→66→96; A[4]: 54. Other buckets empty. Prepend vs append does not affect correctness; slides use prepend.
- **Ops.**
- SEARCH(41): compute h=1, scan list (1→41) → found.
- INSERT(16): already present → no-op.
- DELETE(66): remove from A[6] list.
- **Verification.** Invariants preserved; other buckets unchanged.

✅ **Answer:** Final chaining table has lists A[1]=[41,1], A[4]=[54], A[6]=[96,16] (assuming prepend on insert) and others empty.

Alternative (KT). Any list order is valid; expected list size per slot is $\alpha = n/m$.

Exercise 2 — Chained hashing: time & expected bucket length

Assignment Source: weekplan7.pdf p.1
Text Source: hashing-05.png, hashing-09.png

- **Claim.** Under simple-uniform hashing and $\alpha=n/m$, expected list length at the slot of a fixed key x is $1+(n-1)/m$.
- **Proof (indicators).** Let $I_y=1$ if $h(y)=h(x)$, 0 otherwise. Then $|A[h(x)]|=\Sigma_{y\in S} I_y$. For y=x, $I_x=1$. For $y\neq x$, $E[I_y]=Pr[h(y)=h(x)]=1/m$. Thus $E[|A[h(x)]|] = 1 + (n-1)\cdot(1/m) = 1 + (n-1)/m$.
- **Complexity.** Expected SEARCH/INSERT/DELETE in $O(1+\alpha)$; with $m=\Theta(n)$, this is $O(1)$.

✅**Answer:** $E[|A[h(x)]|] = 1 + (n-1)/m$ and operations run in expected $O(1+\alpha)$.

Exercise 3 — Linear probing: insert/search trace

Assignment Source: weekplan7.pdf p.1
Text Source: hashing-06.png (ops), hashing-08.png (time/variants)

- **Setup.** m=10, h(x)=x mod 10. Insert sequence [41,1,13,54,98] (from slide). Buckets initially empty.
- **Trace.**
- 41 → A[1]=41.
- 1 → A[1] occupied → probe to A[2]=1.
- 13 → A[3]=13.
- 54 → A[4]=54.
- 98 → A[8]=98.
- **Search rule.** Starting at A[h(x)], scan right cyclically until ⊥ or x is found.
- **Verification.** Cluster is the consecutive non-empty run starting at A[1].

✅**Answer:** Final array (indices 0..9): [⊥,41,1,13,54,⊥,⊥,⊥,98,⊥]. SEARCH uses linear scan within the cluster.

Exercise 4 — Lazy deletion in linear probing

Assignment Source: weekplan7.pdf p.2
Text Source: hashing-08.png

- **Why tombstones.** Removing an interior key breaks searches for later keys in the same cluster. Use a special value ◊ (DELETED) that is treated as occupied during SEARCH and as empty during INSERT.
- **Procedure.** On DELETE(x): find j by linear_probe_search; if found, set A[j]←◊. Optionally rebuild when tombstone count exceeds a threshold to keep performance.

✅ **Answer:** Correctness preserved because subsequent keys remain reachable through ◊; periodic rebuild restores probe lengths.

Exercise 5 — Simple uniform hashing: indicator-variable proof

Assignment Source: weekplan7.pdf p.2
Text Source: hashing-09.png

- **Goal.** Show expected chain length equals $1 + (n-1)/m$ (detail in Ex. 2) and hence expected $O(1)$ per operation when $\alpha=\Theta(1)$.
- **Method.** Indicator variables; linearity of expectation; no independence beyond pairwise collision bound needed.

✅**Answer:** Expected chain length $1+(n-1)/m$; expected time $O(1+\alpha)$.

Exercise 6 — Universal hashing: family & collision bound

Assignment Source: weekplan7.pdf p.2
Text Source: hashing-10.png; kt.pdf

- **Family.** For prime $p>|U|$, define $H = \{ h_{a,b}(x) = ((a\cdot x + b) \bmod p) \bmod m \}$ with $a\in\{1,\dots,p-1\}$, $b\in\{0,\dots,p-1\}$.
- **Property.** For any $x\neq y$, $\Pr_{a,b}[h_{a,b}(x)=h_{a,b}(y)] \le 1/m$.
- **Sketch.** Over $\mathbb{Z}_p$, $(a\cdot x+b) \equiv (a\cdot y+b) \Leftrightarrow a(x-y) \equiv 0 \Rightarrow a \equiv 0$ (forbidden) unless $x\equiv y$; for fixed (x,y), at most one b matches per a, giving $\le 1/p$; reduction mod m preserves $\le 1/m$ when $m\le p$.
- **Practice.** Choose a,b uniformly on (re)build; guarantees expected $O(1)$ per op independent of input S.

✅ **Answer:** The family above is universal; collision probability $\le 1/m$; with $m=\Theta(n)$ we obtain $O(1)$ expected per operation.

Alternative (dot product). Represent x in base m as vector and use $h_a(x)=(a\cdot x) \bmod m$; H is universal (slides).

Exercise 7 — Puzzle: "Billy & the carrots"

Assignment Source: weekplan7.pdf p.2
Text Source: — (needs exact statement)

- **BLOCKER — need the precise model.** The plan mentions Billy "might go to the same bush again in the next round." To compute E[visits until three carrots], we must know: 1) number of bushes B; 2) how many bushes contain carrots (exactly 3, or each has probability p of having a carrot per visit?); 3) with/without replacement after finding a carrot; 4) whether Billy avoids revisiting successful bushes.
- **If** each visit is an independent Bernoulli(p) success and carrots are unlimited, then E[T to 3 successes] = $3/p$ (negative binomial).
- **If** exactly three distinct bushes hide carrots among B and Billy samples bushes **with replacement**, the process is a coupon-collector variant; expected time depends on revisits and

equals $\Sigma_{i=0}^{2} 1/((3-i)/(B))$ after successful-bush avoidance. Please paste the exact statement to finalize.

✅**Answer:** Pending exact statement; see cases above.

Puzzle (pick-one for practice)

Design a universal family. For 32-bit integers and table size $m=2^k$ ($k \leq 16$), propose a fast $h(x)$.
Hint: multiply-shift: choose odd 32-bit A uniformly; return $(A \cdot x) \gg (32-k)$. This is 2-universal and branch-free.

Summary

- **What to use when.**
- Use **chaining** when you want simple deletion and predictable $O(1+\alpha)$ behavior.
- Use **linear probing** for cache locality; keep α low and rebuild when tombstones accumulate.
- Use **universal hashing** (e.g., multiply-shift or ax+b mod p) to decouple performance from adversarial S.
- **Resize policy.** Maintain α in [0.5, 0.75]; double/halve m and rehash with a fresh h.
- **Notation recap.** $\alpha=n/m$ (load factor); δ (delta) bottleneck when used in flows (not here); U universe; S stored set.

Next actions. Paste a screenshot of the exercise block in weekplan7.pdf so we can lock exact labels/IDs and finalize Exercise 7.