**Lecture 10: Graph**

**Shortest Path**

Shortest path between vertices u and v – The path where the weighted sum of edges becomes the minimum, among the paths that connects u and v. – Weight: cost, distance, or time – Example Shortest path problem from vertex 0 to vertex 3 • The weight of a node pair without an edge in the adjacent matrix is ∞. • Shortest path: 0, 4, 1, 2, 3 • The shortest path length: 3 + 2 + 4 + 2 = 11

SSP: **Bellman-Ford** algorithm, Dijkstra algorithm

ASP: **Floyd-Warsha**ll algorithm, Johnson algorithm

**Variants of Single-Source Shortest Path**

- Single-source shortest path (SSP)

: Find a shortest path to each vertex t from a given source vertex v

- Single-destination shortest path

: Find a shortest path to a given destination vertex t from each vertex v – When reversing the edge direction in the graph, it becomes the SSP problem

- Single-pair shortest path

: Find a shortest path from u to v for given vertices u and v .Sub problem of the single source shortest path

-All-pairs shortest paths (ASP)

: Find a shortest path from u to v for every pair of vertices u and v. Running single source for each vertex ◊ Too slow .There is a better (faster) algorithm for this! (e.g., using dynamic programming)

**Dijkstra Algorithm**

**:** Find the shortest path from one vertex to all other vertices

**텍스트, 사람, 실내, 벡터그래픽이(가) 표시된 사진

자동 생성된 설명**

**◾** Key idea

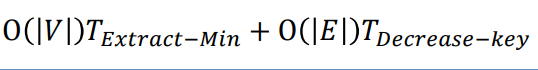
– For G = (V,E), select the vertex u with the minimum shortest path estimate from the priority queue Q=V-S

– Then, perform relaxation (update) for all edges of vertex u

**◾**Similar to BFS

**◾**similar to Prim algorithm for MST

**◾**Time Complexity

:****

**Topological Sort**

**:** List all vertices according to the precedent relation on the directed acyclic graph (DAG)

**테이블이(가) 표시된 사진

자동 생성된 설명**

**Lecture 11: Hashing**

**:** Search operation is based on the comparison with key value – Finds the item to be searched by comparing the key with saved items – Time complexity: ܱ(݊) for unsorted data, ܱ(݈݋݃ଶ݊) for sorted data. Hashing Computes the item address on the hash table by an arithmetic operation on the key value, and then accesses the item . Yime complexity: O(1) ideally ,Hashing is similar to organizing things

**Abstract Data Type of Dictionary Structure**

**◾** Dictionary structure

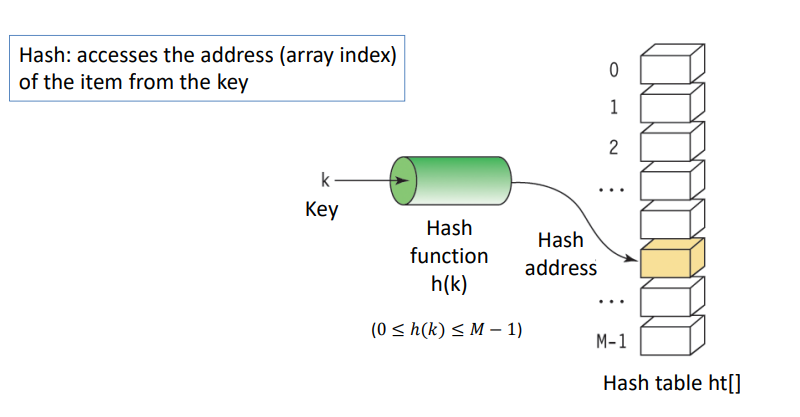
– Called as map or table

– Consists of two fields associated with search. -> key, Value

**Hashing Structure**

**:** Generate hash address using search key

- Hash address: index of the hash table implemented as an array.

****

**Hash Table**

**:** Table with M buckets

**Ideal Hashing**

Example: save and search student data with hashing

– Student ID: 5 digits (2 digits for department and 3 digits for student number)

– For students from the same department, use only the last 3 digits.

– When student ID: 00023, the student's information is stored in ‘ht[23]

’ – If the hash table has 1000 spaces, the search time is O(1), which is ideal.

• No collision (or overflow) occurs.

**Hashing in Practice**

**:** In practice, the size of the hash table is limited So, you can not allocate storage space for all possible keys. Usually, Hash table size << # of possible keys

**Hash Function**

Condition for hash function

– Fewer collisions

– The hash function values should be distributed within the address space of the hash table as evenly as possible.

– Fast computation

Type of hash function

– Division, folding, median-square, bit extraction, numerical analysis

1. Division : ℎ(k) = k mod M -> The size ܯ of the hash table is a prime number
2. Folding : Shift folding, boundary folding, and XOR folding

**텍스트이(가) 표시된 사진

자동 생성된 설명**

1. Median-square function : Squares the search key, then takes a few bits to generate a hash address
2. Bit extraction function : Hash table size: ܯ = 2 ௞ – Considering the search key as a binary number, use k bits at an arbitrary position as a hash address
3. Numerical analysis method

– Consider the distribution property of digits of key

– Combine keys that are evenly distributed according to the size of the hash table

**Collision**

**:** Items with different search keys have the same hash address . Cannot save items in hash table when collision occurs (for a single slot)

**Solutions to collision**

– Probing: stores conflicted items in a different location of the hash table (linear probing and quadratic probing) – Chaining: uses the linked list in each bucket

**Linear Probing**

**테이블이(가) 표시된 사진

자동 생성된 설명**

**텍스트, 캐비닛, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

**Quadratic Probing**

**:** ℎ(k) = k mod M is used

Clustering and coalescing, which is a problem in linear probing, can be greatly mitigated.

**Double Hashing**

**:** Also known as rehashing. When collision occurs, it uses a different hash function than the original hash function.

**Chaining**

**:** Address collision and overflow issues with linked lists

– Each bucket is not assigned a fixed slot, but a linked list that is easy to insert and delete – Sequential search in the linked list for each bucket

**Performance Analysis of Hashing**

**테이블이(가) 표시된 사진

자동 생성된 설명**