**해시테이블 개요**

* **해시테이블(hash table)**: 키-주소 매핑에 의해 구현된 사전 ADT이다.
  + 예시: 컴파일러의 심볼 테이블, 환경변수들의 레지스트리.
* 해시테이블은 **버켓 배열**과 **해시함수**로 구성된다.
  + 항목들의 키를 주소(즉, 배열 첨자)로 매핑하여 1차원 배열에 사전 항목들을 저장하는 구조이다.

**해시테이블의 성능**

* **성능**:
  + 탐색, 삽입, 삭제의 최악 시간 복잡도는 \*\*O(n)\*\*이다.
  + 그러나 기대 시간 복잡도는 \*\*O(1)\*\*로 매우 빠르다.

**버켓 배열 설명**

* 해시테이블을 위한 \*\*버켓 배열(bucket array)\*\*은 크기 **M**의 배열 **A**로 구성된다.
  + A의 각 셀은 **버켓**으로 간주되며, 슬롯(slot)이라고도 불린다.
  + 정수 **M**은 배열의 용량을 정의한다.
  + 키 **k**를 가진 원소 **e**는 버켓 \*\*A[k]\*\*에 삽입된다.
  + 사전에 존재하지 않는 키에 속하는 버켓 셀들은 **NoSuchKey**라는 특별한 개체를 담는 것으로 가정한다.

**해시함수 및 충돌 해결**

* **해시함수(hash function)** **h**: 주어진 형의 키를 고정 범위 \*\*[0, M – 1]\*\*로 매핑한다.
  + 예시: **h(x) = x % M**.
* 주어진 키 형의 해시테이블은 다음 두 가지로 구성된다:
  + 해시함수 **h**.
  + 크기 **M**의 배열(테이블이라 불림).
* 사전을 해시테이블로 구현할 때, 목표는 항목 \*\*(k, e)\*\*를 첨자 \*\*i = h(k)\*\*에 저장하는 것이다.

**충돌 해결 방법**

* **충돌(collision)**: 두 개 이상의 원소들이 동일한 셀로 매핑되는 경우를 의미한다.
  + 즉, 상이한 키 **k1**과 **k2**에 대해 \*\*h(k1) = h(k2)\*\*이면 충돌이 발생했다고 말한다.
* 충돌 해결을 위한 일관된 전략이 필요하다.

**개방주소법 설명**

* **개방주소법(open addressing)**: 충돌 항목을 테이블의 다른 셀에 저장하는 방법이다.
  + 장점: 분리연쇄법에 비해 공간 사용을 절약할 수 있다.
  + 단점: 삭제가 어렵고, 사전 항목들이 연이어 군집화(clustering)될 수 있다.

**해시테이블의 재해싱**

* **재해싱**: 해시테이블의 적재율을 상수(보통 0.75) 이하로 유지하기 위해 원소를 삽입할 때마다 추가적인 작업이 필요하다.
  + 재해싱을 수행하는 시점:
    1. 적재율의 최적치를 초과했을 때.
    2. 삽입이 실패한 경우.
    3. 너무 많은 비활성 셀들로 포화되어 성능이 저하되었을 때.

**그래프 개요**

* **그래프(graph)**: 정점(vertex)과 간선(edge)으로 구성된 데이터 구조이다.
  + 정점은 노드의 집합을 의미하며, 간선은 정점 쌍의 집합을 의미한다.
* 그래프는 다양한 정보를 저장하는 데 사용된다.
  + 예시: 공항을 표현하는 그래프에서 정점은 공항, 간선은 항로를 나타낸다.

**그래프의 주요 개념**

* **정점과 간선**: 그래프의 기본 구성 요소로, 정점은 정보를 저장하고 간선은 정점 간의 관계를 나타낸다.
* **정점의 차수(degree)**: 정점에 연결된 간선의 수를 의미한다.
* **경로(path)**: 정점과 간선의 교대열로, 정점으로 시작하여 정점으로 끝난다.

**최소신장트리 개념**

* **최소신장트리(minimum spanning tree, MST)**: 가중그래프의 총 간선 무게가 최소인 신장트리이다.
  + 신장 부그래프는 그래프의 모든 정점들을 포함하는 부그래프를 의미한다.
* MST는 통신망, 교통망 등 다양한 분야에 응용된다.

**Prim-Jarnik 알고리즘 설명**

* **Prim-Jarnik 알고리즘**: 단순 연결 무방향 가중그래프에서 MST를 찾기 위한 탐욕 알고리즘이다.
  + 임의의 정점 **s**를 선택하여 시작하며, 배낭 안에서 MST를 키워 나간다.
  + 각 정점에 대해 거리 라벨을 정의하고, 반복적으로 최소 거리의 정점을 선택하여 MST에 추가한다.

**Kruskal 알고리즘 설명**

* **Kruskal 알고리즘**: MST를 찾기 위한 또 다른 탐욕 알고리즘이다.
  + 모든 정점을 각각의 독자적인 배낭에 넣고, 간선을 우선순위 큐에 저장한다.
  + 반복적으로 최소 무게의 간선을 선택하여 MST에 추가하고, 배낭을 합친다.

**Baruvka 알고리즘 개요**

* **Baruvka 알고리즘**: Kruskal이나 Prim-Jarnik 알고리즘과 달리, 여러 개의 간선을 동시에 선택하여 MST를 찾는 방법이다.
  + 각 연결 요소에서 최소 무게의 간선을 선택하여 MST를 구성한다.

**최단경로 문제 정의**

* **최단경로 문제(shortest path problem)**: 가중그래프와 두 개의 정점 **u**와 **v**가 주어졌을 때, u와 v 사이의 무게의 합이 최소인 경로를 구하는 문제이다.
  + 최단경로의 길이는 간선들의 무게 합으로 정의된다.

**Dijkstra 알고리즘 설명**

* **Dijkstra 알고리즘**: 출발정점 **s**로부터 다른 모든 정점까지의 거리를 계산하는 알고리즘이다.
  + 그래프가 연결되어 있고, 간선의 무게가 음수가 아닌 경우에 적용된다.

**Bellman-Ford 알고리즘 설명**

* **Bellman-Ford 알고리즘**: 음의 무게를 가진 간선이 있는 경우에도 작동하는 최단경로 알고리즘이다.
  + 모든 간선을 n-1번 완화하여 최단경로를 찾는다.

**모든 쌍 최단경로 문제**

* **모든 쌍 최단경로(all-pairs shortest paths) 문제**: 가중 방향그래프 G의 모든 정점 쌍 간의 거리를 찾는 문제이다.
  + Dijkstra 알고리즘을 n번 호출하거나, Bellman-Ford 알고리즘을 n번 호출하여 해결할 수 있다.

**응용 문제: 항공편 스케줄링**

* 항공편 스케줄링 문제는 주어진 두 개의 공항 **a**와 **b** 및 시각 **t**에 대해, a에서 t 정시 혹은 이후에 출발할 경우 가장 이른 시각에 b에 도착할 수 있도록 하는 연결 항공편을 계산하는 것이다.

**응용 문제: 괴물성에 갇힌 낙랑**

* 호동이 괴물성에 갇힌 낙랑을 구출하기 위한 문제로, 미로의 구조는 방과 복도로 구성된 방향그래프이다.
* 각 방의 상태에 따라 에너지를 관리하며, 에너지를 0 이상으로 유지해야 한다.