CSE 326: Data Structures

Hashing

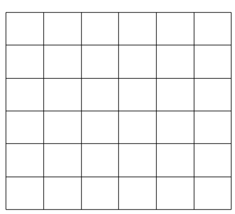
Neva Cherniavsky
Summer 2006

Announcements

- Midterms
  - Gary will hand out tomorrow
- Project Phase C due tomorrow
  - Brief overview of Kruskal's method today

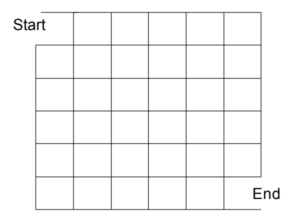
Cute Application

- Build a random maze by erasing edges.



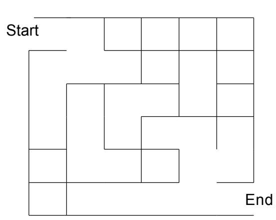
Cute Application

- Pick Start and End



Cute Application

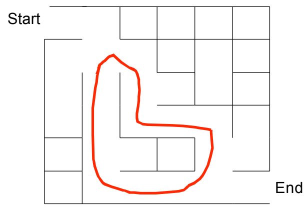
- Repeatedly pick random edges to delete.



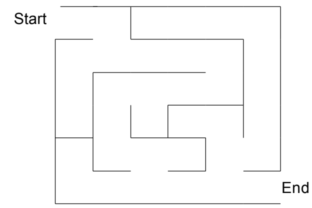
Desired Properties

- None of the boundary is deleted
- Every cell is reachable from every other cell.
- There are no cycles – no cell can reach itself by a path unless it retraces some part of the path.

A Cycle



A Good Solution



Number the Cells

We have disjoint sets S ={ {1}, {2}, {3}, {4},… {36} } each cell is unto itself.
We have all possible edges E ={ (1,2), (1,7), (2,8), (2,3), … } 60 edges total.

Start	1	2	3	4	5	6	
	7	8	9	10	11	12	
	13	14	15	16	17	18	
	19	20	21	22	23	24	
	25	26	27	28	29	30	
	31	32	33	34	35	36	End

Basic Algorithm

- S = set of sets of connected cells
- E = set of edges
- Maze = set of maze edges initially empty

```
While there is more than one set in S
  pick a random edge (x,y) and remove from E
  u := Find(x);
  v := Find(y);
  if u ≠ v then
    Union(u,v)
  else
    add (x,y) to Maze
All remaining members of E together with Maze form the maze
```

Example Step

Pick (8,14)

Start	1	2	3	4	5	6	
	7	8	9	10	11	12	
	13	14	15	16	17	18	
	19	20	21	22	23	24	
	25	26	27	28	29	30	
	31	32	33	34	35	36	End

S
{1,2,7,8,9,13,19}
{3}
{4}
{5}
{6}
{10}
{11,17}
{12}
{14,20,26,27}
{15,16,21}
.
.
{22,23,24,29,30,32
33,34,35,36}

Example

S
{1,2,7,8,9,13,19}
{3}
{4}
{5}
{6}
{10}
{11,17}
{12}
{14,20,26,27}
{15,16,21}
.
.
{22,23,24,29,39,32
33,34,35,36}

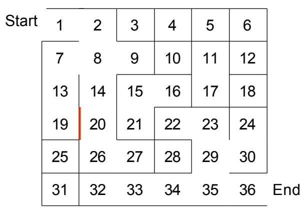
Find(8) = 7
Find(14) = 20

Union(7,20)

S
{1,2,7,8,9,13,19,14,20 26,27}
{3}
{4}
{5}
{6}
{10}
{11,17}
{12}
{15,16,21}
.
.
{22,23,24,29,39,32
33,34,35,36}

Example

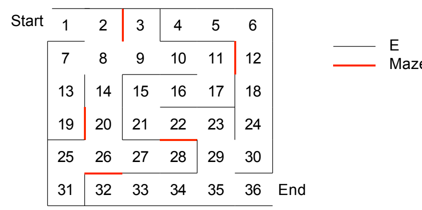
Pick (19,20)



S
{1,2,7,8,9,13,19
14,20,26,27}
{3}
{4}
{5}
{6}
{10}
{11,17}
{12}
{15,16,21}
.
.
{22,23,24,29,39,32
33,34,35,36}

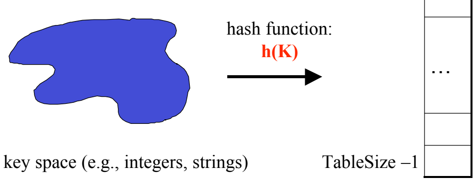
Example at the End

S
{1,2,3,4,5,6,7,… 36}



Hash Tables

- Constant time accesses!
- A **hash table** is an array of some fixed size, usually a prime number.
- General idea:



Simple Hash Table

T

0	
1	
2	John Smith
3	
4	Judy Jones
5	
6	Martha Lee
7	Jerry Lee
8	
9	

Hash function:

h : U → { 0,1,…,Hsize -1}

U is the universe of keys

h("name") is the hash value of "name"

h(Judy Jones) = 4
h(Jerry Lee) = 7

Find("name") = T[h("name")]

Example

- key space = integers
- TableSize = 10
- **h**(K) = K mod 10
- **Insert**: 7, 18, 41, 94

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Another Example

- key space = integers
- TableSize = 6
- **h**(K) = K mod 6
- **Insert**: 7, 18, 41, 34

0	
1	
2	
3	
4	
5	

Student Activity

General Idea

- Key space of size M, but we only want to store subset of size N, where N<<M.
  - Keys are identifiers in programs. Compiler keeps track of them in a symbol table.
  - Keys are student names. We want to look up student records quickly by name.
  - Keys are chess configurations in a chess playing program.
  - Keys are URLs in a database of web pages.

Hash Functions

1. **simple/fast** to compute,
2. Avoid **collisions**
3. have keys distributed **evenly** among cells.

Time for insert/delete/find?

Downsides?

Sample Hash Functions:

- key space = strings
- $s = s_0\ s_1\ s_2 \ldots s_{k-1}$

Pluses and minuses?

TableSize?

1. $h(s) = s_0$ mod TableSize
2. $h(s) = \left(\sum_{i=0}^{k-1} s_i\right)$ mod TableSize
3. $h(s) = \left(\sum_{i=0}^{k-1} s_i \cdot 37^i\right)$ mod TableSize

Designing a Hash Function for web URLs

$s = s_0\ s_1\ s_2 \ldots s_{k-1}$

Issues to take into account:

h(s) =

Good Hash Functions

- Integers: Division method
  - Choose Hsize to be a prime (Why?)
  - h(n) = n mod Hsize
  - Example. Hsize = 23, h(50) = 4, h(1257) = 15
  - When might this fail?

Good Hash Functions

- Character Strings
  - $x = a_0a_1a_2 \ldots a_m$ is a character string. Define

    $int(x) = a_0 + a_1 128 + a_2 128^2 + \ldots + a_m 128^{m-1}$

    h(x) = int(x) mod Hsize
  - Compute h(x) using Horner's Rule

```
h :=0
for i = m to 0 by -1 do h := (a_i +128h) mod Hsize
return h
```

tableSize: Why Prime?

- Suppose
  - data stored in hash table: 7160, 493, 60, 55, 321, 900, 810
  - tableSize = 10
    data hashes to 0, 3, 0, 5, 1, 0, 0
  - tableSize = 11
    data hashes to 10, 9, 5, 0, 2, 9, 7

Real-life data tends to have a pattern

Being a multiple of 11 is usually *not* the pattern ☺

A Bad Hash Function

- Keys able1, able2, able3, able4
  - Hsize = 128
    int(ablex) mod 128 = int(a) = 97
    Thus, h(ablex) =h(abley) for all x and y

What is the central problem we're trying to avoid?

How can we fix it?