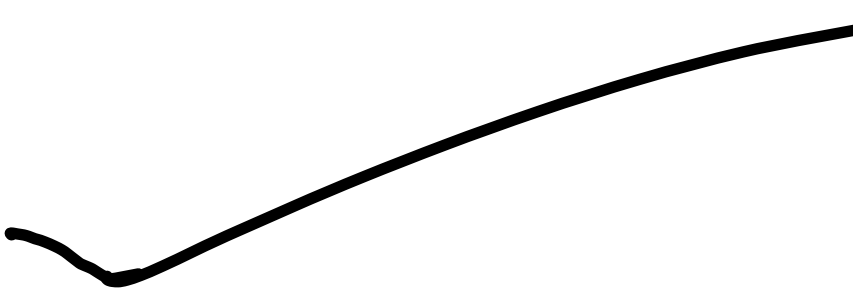
ADSA Revision



Integer Arithmetic

week 1

School method of Addition.

School method of Multiplication

Representation :

Assume that integers are represented as digit strings.

Base B number system, $B > 1$

Digits $0, 1, \cdots, B-1$

$a_{n-1} a_{n-2} \cdots a_1 a_0$

store in an array a of size n containing n digits.

represents $\sum_{i=0}^{n-1} a_i B^i$

Theorem:

The addition of two n-digit integers requires exactly n primitive operations. The result is an n+1 digit integer.

School Method for addition

Input: Two integers $a = (a_{n-1} \cdots a_0)$ and $b = (b_{n-1} \cdots b_0)$

Compute: $S = a + b$,

where $S = (S_n \cdots S_0)$ is an n+1 digit integer.

$a_{n-1} \cdots a_1 a_0$		first operand
$b_{n-1} \cdots b_1 b_0$		second operand
C_n	$C_{n-1} \cdots C_1$ 0	carries
S_n	$S_{n-1} \cdots S_1 S_0$	sum

$C_n \cdots C_0$ is sequence of carries

$C_0 = 0$, $C_{i+1} \cdot B + S_i = a_i + b_i + c_i, (0 \le i < n)$, $S_n = C_n$

Pseudo-code:

```
c = 0 ;
for (i = 0 , i < n , i++)
{
  add a_i, b_i and c to form s_i and a new carry c ;
}
s_n := c
```

Example for Multiplication

$a = 342$, $b = 26$, $B = 10$

1. Compute partial products

$a \cdot b_1 = 342 \times 2$ $\qquad$ $a \cdot b_0 = 342 \times 6$

$p_1 = 684$ $\qquad$ $p_2 = 2052$

2. Sum up aligned partial products

$$\begin{array}{r} 6840 \\ +\ 2052 \\ \hline 8892 \end{array}$$

School Method for Multiplication

Input: Two integers $a = (a_{n-1} \ldots a_0)$ and $b = (b_{n-1} \ldots b_0)$

Compute: $p = a \cdot b$, where $p = (p_{2n-1} \ldots p_0)$ is a $2n$ digit-long number.

number of primitive operations:

n multiplications, $n+1$ additions, $2n+1$ in total.

Example: $a = 342$, $b_j = 6$, $B = 10$ 基底 Base.

computing c's and d's

$a_0 \cdot b_j = 2 \cdot 6 = 12$

$c_0 = 1$, $d_0 = 2$

$a_1 \cdot b_j = 4 \cdot 6 = 24$

$c_1 = 2$, $d_1 = 4$

$a_2 \cdot b_j = 3 \cdot 6 = 18$

$c_2 = 1$, $d_2 = 8$

summing up.

$$\begin{array}{r} 1210 \\ +\ 842 \\ \hline 2052 \end{array}$$

Pseudo-code

```
p = 0;
for (j = 0; j < n; j++)
{
    p = p + a · b_j · B^j;
}
```

Theorem

The school method multiplies two n-digit integers with $3n^2 + 2n = \Theta(n^2)$ primitive operations

week 2

Recursive Version of the School Method

Karatsuba Multiplication.

(Divide and conquer)

- Divide the problem into sub-problems,
- Solve sub problems using same approach.
- Obtain solution for original problem from solution to sub solution.

$$a \cdot b = a_1 \cdot b_1 \cdot B^{2k} + (a_1 \cdot b_0 + a_0 \cdot b_1) \cdot B^k + a_0 \cdot b_0$$

Algorithm:

1. Split a and b to obtain a_1, a_0, b_1 and b_0
2. Compute $a_1 \cdot b_1$, $a_1 \cdot b_0$, $a_0 \cdot b_1$, and $a_0 \cdot b_0$
3. Add the aligned products to obtain $p = a \cdot b$.

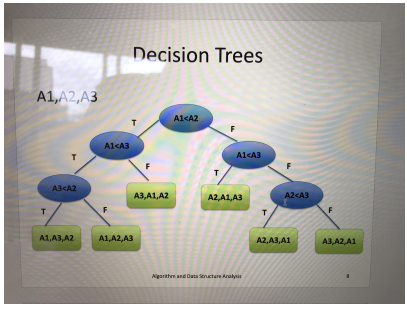
Linear-Time Sorting Algorithm (Counting sort)

	Insertion Sort	Merge Sort	Heap Sort	Quick Sort
worst case:	$O(n^2)$	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
average case:	$O(n^2)$			
best case:	$O(n)$	$O(n \log n)$		$O(n \log n)$

Theorem: all comparison sorts are $\Omega(n \log n)$

Decision Tree : provide an abstraction of comparison sorts

eg.



Any decision tree that sorts n elements has height $\Omega(n \log n)$.

Skip List

Lecture 12 week 7

Runtime of search: $O(\log n)$

Height of skip list: $O(\log n)$

# of pointers $O(2n + \lceil \log n \rceil + 3)$

关键动作:

Search ( )

Split ( ) 分割一个 skip list

Concatenate ( ) 聚合两个 skip list.

Delete ( )

Insert ( )

Space Usage: The expected space usage of a skip list with n items is $O(n)$.

Height: a skip list with n entries has height at most $O(\log n)$ with probability at least $1 - \frac{1}{n^{c-1}}$ that is asymptotically 1 for large constant c.

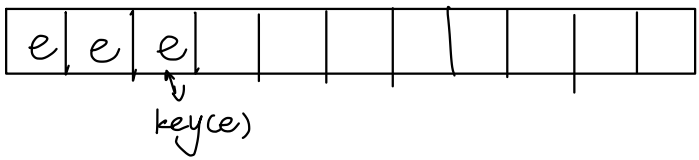
Hashing (1)

Lecture 13 week 7

Worst case analysis of data structures:

Name:	Insert (x)	Remove(x)	Find(x)
Linked List:	$O(1)$	$O(1)$	$\Theta(1)$
AVL Trees	$O(\log n)$	$O(\log n)$	$O(\log n)$

Associative Arrays 关联数组

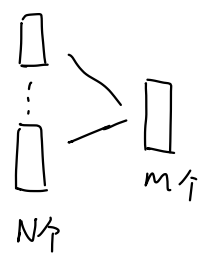


Hash table

Idea: use hash function h to map potential N keys to m values, where $m < N$.

Let t be a hash table of size m.

Store element e in index $h(key(e))$ of it.



worst case: hash function of elements returns the same value. e.g. insert zebra, boa, ABBA, cobra.

通常来说: $n < \frac{N}{m}$

- n: number of elements
- N: number of potential keys
- m: number of possible hash function values

但: it is possible to have all n elements in one table entry.

Insert(e)

1. Get index $h(key(e))$
2. Add e to the end of the list at $t[h(key(e))]$ — $O(1)$

∴ worst case insert of linked list is $O(1)$

∴ Insert(e: Element) is $O(1)$

$O(1)$

Find (k : Key)

1. Get index h(k)
2. Search through list at t[h(k)] $\Theta(n)$
3. If element e with unique key k is in list, return e.
   Else return null. $O(1)$

∵ Worst case find of linked list is $\Theta(n)$
∴ Find (k:key) is $\Theta(n)$

Remove (k: Key)

1. Get index h(k)
2. Search through list at t[h(k)]
3. If element e with unique key k is in list, remove e. $O(1)$

∵ worst case find of linked list is $\Theta(n)$
worst case remove of linked list is $O(1)$
∴ remove (k: Key) is $\Theta(n)$

Average Case Analysis

Theorem 4.1 If n elements are stored in a hash table t with m entries and a random hash function is used, the expected execution time of remove or find is O(1+n/m).

Theorem 4.2 Let c be a positive constant. A family H of functions from Key to 0..m-1 is called c-universal if any two distinct keys collide with a probability of at most c/m.

Theorem 4.3 If n elements are stored in a hash table with m entries using hashing with chaining and a random has function from a c-universal family is used, the expected execution time of remove or find is O(1+cn/m).

Hashing (2)

week 8

1 universal family 是什么东西?

Alterative Approach to Hashing

Hashing with chaining is a closed hashing approach.

- closed hashing: handles collision by storing all elements with the same hashed key in one table entry.
- open hashing: handles collision by storing subsequent elements with the same hashed key in different table entries.

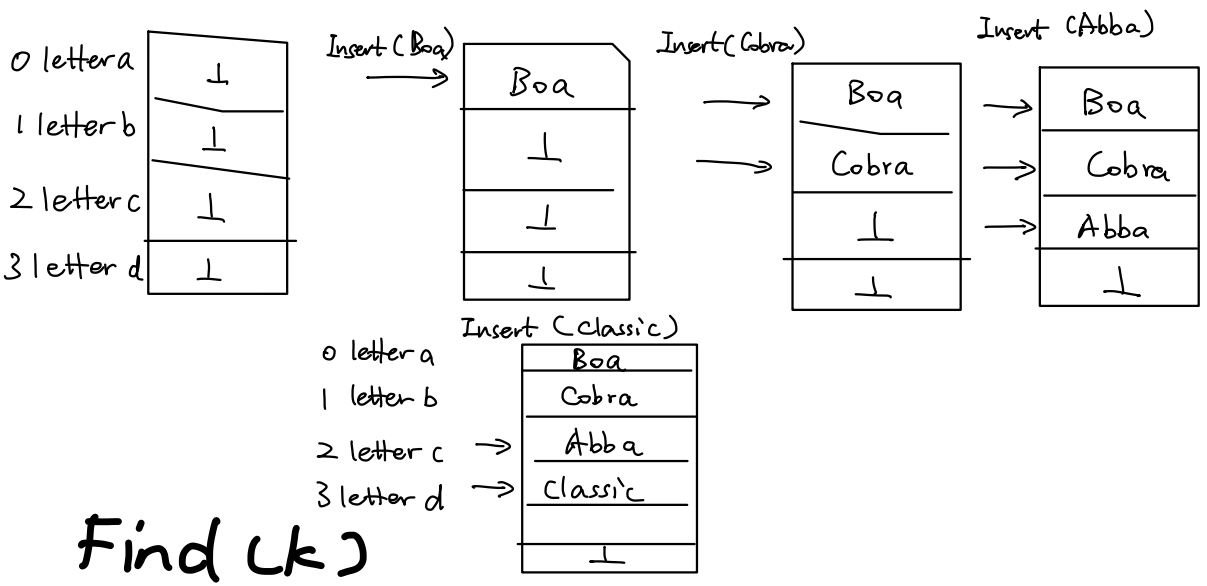
Hashing with linear Probing (open hashing)

All unused entries in t are set to $\perp$

When inserting on a collision, insert the element to the next free entry.

Insert(e: Element)

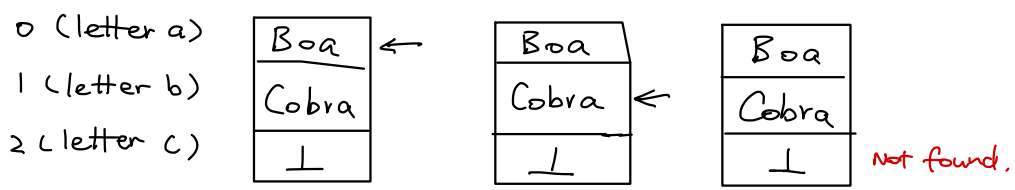
1. get index i = h(key(e))
2. if t[i] == $\perp$, store e at t[i] 只有在t[i]是空的时候才可以insert.
3. if t[i] is not empty, increase i by 1 and go to step 2.



Find (k)

1. Get index $i = h(k)$
2. If $t[i] == \perp$, return not found
3. If element e at $t[i]$ has key (e) == k, return found.
   Else increase i by 1 and go to step 2.

e.g. Find (ABBA)



Remove (k)

Cannot remove the element with key(e)==k and replace it with NULL.
If we replace element e1 at t[i] with NULL, how do we find an element e2 with the same h(k)

Instead, first remove the element with key(e) ==k and then fix the invariant.

Sudo Code:

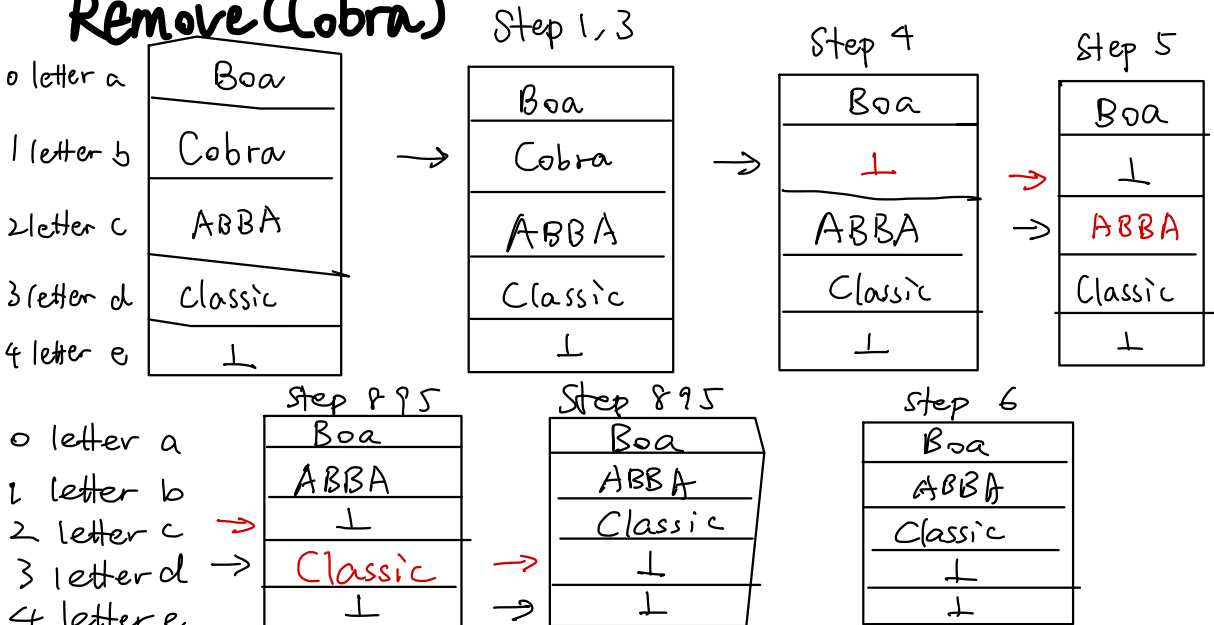
search (k)

1. Get index i=h(k)
2. If t[i] == NULL, return
3. If element e at t[i] has key(e) != k, increase i by 1 and go to step 2.
4. Set t[i] = NULL

repair

5. Set index j=I+1
6. If t[j] ==NULL, return
7. If h(t[j]) > I, increase j by 1
8. Else set t[i] = t[j] and t[j] = NULL
9. Set i = j and go to step 5

Remove(Cobra)



	Chaining	VS. Linear Probing
pro	referential integrity	use of contiguous memory
con	waste of space	gets slower as table fills up

Graphs - 1

week 8

A directed graph (digraph) G=(V,E) is a pair consisting of a node set (vertex set) V and An edge set (arc set) E⊆V×V

$n = |V|$ the # of vertices $m = |E|$ the # of edges

$C: E \to R$ edge weights/costs

Terminology 术语

An edge $e = (u, v)$ represents a connection from u to v.

incident to u & v; source; target

u & v are adjacent

* Edge (v, v) is called a self-loop.

- The number of outgoing edges of a vertex v is called the out-degree of v : outdegree(v) = $|\{(v, u) \in E\}|$
- The number of incoming edges of a vertex v is called the indegree of v: indegree(v) = $|\{(u, v) \in E\}|$

Bidirected graph : $(u,v) \in E \Rightarrow (v,u) \in E$

Undirected graph : streamlined representation of a bidirected graph. (改进的)

we write (u,v) and (v,u) as a two element set $\{u,v\}$

Subgraph

Paths :

Simple Graph Algorithm

Observation: node with outdegree zero can not appear in a cycle .

An undirected graph is called a tree if there is exactly one path between any pair of nodes.

An undirected graph is called a forest if there is at most one path between any pair of nodes.

Note that each component of a forest is a tree.

Two choices for representation:

1. Adjacency Lists

use for each node v a double-linked list that stores its outgoing neighbours (alternatively we can also use the incoming neighbours or lists for both.

2. Adjacency Matrices

Represent a graph consisting of n nodes by an $n \times n$ matrix A. Set

$A_{ij} = 1$ if $(i,j) \in E$

$A_{ij} = 0$ otherwise.

Advantage

- Insertion of edges goes in constant time
- well suited for sparse graphs

Insertion, removal, edge queries work in constant time.

O(n) to obtain an edge entering or leaving a node.

Disadvantage

storage requirement n^2 even for sparse graphs.

Graph Traversal 遍历 到每个 node

breath-first-search 广度优先

use Adjacency array and Priority Queues &.

Total Runtime: $O(n+m)$

Pseudo-code

Cheat Sheet ※- 抄!

Depth first Search 深度优先搜索

Idea for DFS
Whenever you visit a vertex, explore in the next step one of its non-visited neighbors.

Implementation
When visiting a node, mark it as visited and recursively call DFS for one of its non-visited neighbors.
If there is no non-visited neighbor end recursive call.

Order in which nodes are finished.

DFS Cheat Sheet

必抄！

Runtime: O(m+n)

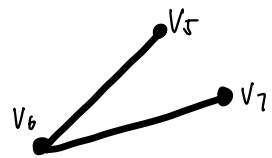
Strongly connected components

Two nodes u and v belong to the same strongly connected component if there is a path from u to v and a path from v to u.

↑ undirected graph always say "yes"!

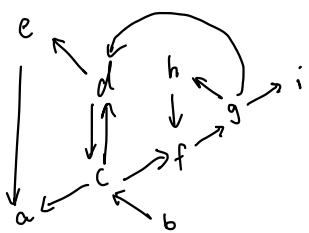
有从 u 去 v 的路也有从 u 去 v 的路.

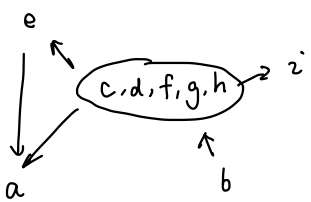
Undirected Graph



$\{v_5, v_6, v_7\}$

Directed Graph





Runtime

- Use adjacency lists to represent the directed graph.
- We use DFS twice (time $O(m+n)$)
- Have to compute the transpose graph (time $O(m+n)$)
- Total runtime: $O(m+n)$

Shortest Path

week 8

single Source Shortest Paths Problem 单源最短路径问题

① Dijkstra 单源最短路径算法，时间复杂度为 O(E+VlogV)，要求权值非负.

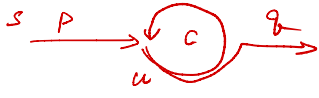
② Bellman-Ford 单源最短路径算法，时间复杂度为 O(VE)，适用于带负权值情况.

All-pairs Shortest Paths Problem：即分别以每个顶点作为源顶点并求其至其它顶点的最短距离.
全源最短路径问题.

Single source shortest path problem:
Compute for a given node s of V a shortest path to any other node in V.

If a path from s to v contains a negative cycles then a shortest path does not exist (is not defined)

有 negative cycle 存在那么就不存在最短路径



命题：最短路径中的子路径也一定是最短的. ✓

Dijkstra's Algorithm

works for acyclic graphs and for non-negative edge cost.

https://youtu.be/pVfj6mxhdMw

Runtime:

Original: Insert and decreaseKey take time $O(1)$
Delete Min takes time $O(n)$
Total Runtime: $O(m+n^2)$

Improvement: Binary Heaps: $O((m+n)\log n)$
Fibonacci Heaps: $O(m+n\log n)$

Bellman-Ford Algorithm

Solves the problem for arbitary edge costs.
任意 边长度.

可以有负权边，不能有负权环. Bellman-Ford 算法可判断图中是否有负权环.

如果一个图没有负权环，从一点到另外一点的最短路径，最多经过所有的V个顶点，有V-1条边

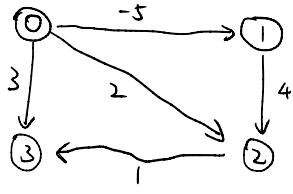
否则，有在顶点经过了两次，既存在负权环.

松弛操作：尝试经由一个点，再回到原来的点，得到更短的路径，多一条边，权值更小.

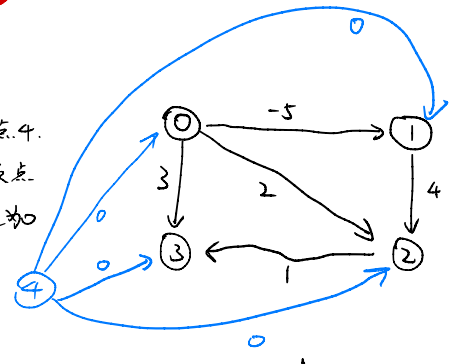
对所有的点进行 V-1 次松弛操作，理论上就找到了从源点到其它所有点的最短路径. 如果还可以继续松弛，所说原图中有负权环.

All-pairs Shortest Paths Problem

week 10

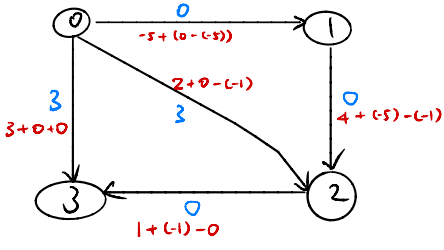


首先，新增一个源顶点4，并使其与所有顶点连通，新边赋权值为0



使用 Bellman-Ford 算法计算新的顶点到其它顶点的最短路径，则从4至[0,1,2,3]的最短路径分别是{0，-5，-1，0}，当得到这个h[]信息后，将新增的顶点4移除，然后使用如下公式对所有权的权值进行re-weight.

$$w[u,v] = w[u,v] + (h[u] - h[v])$$



此时所有权值都为非负，就可以用 Dijkstra 算法对每个顶点进行最短路径计算了。

Runtime: $O(V^2 \log V + VE)$

Floyd-Warshall Algorithm

针对稠密图的 Floyd- Warshall算法 时间复杂度为 $O(V^3)$

2种可能结果：

① The node k+1 does not improve the shortest path from i to j.

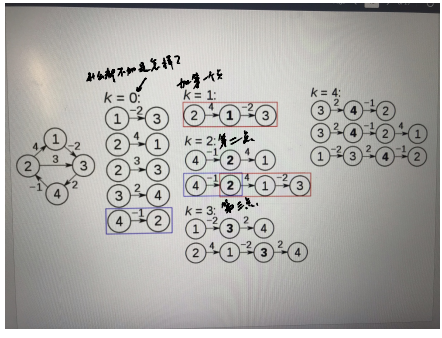
② We get a shorter path by going from i to k+1 and from k+1 to j.

Take the best option, we get:

$$d_{k+1}(i,j) = \min\{d_k(i,j),\ d_k(i,k+1) + d_k(k+1,j)\}.$$ ✓

The length of a shortest path from i to j in given Graph G is $d_{n,n}(i,j)$

e.g.



The distance matrix at each iteration of k, with the updated distances in **bold**, will be:

$k=0$

i \ j	1	2	3	4
1	0	∞	−2	∞
2	4	0	3	∞
3	∞	∞	0	2
4	∞	−1	∞	0

$k=1$

i \ j	1	2	3	4
1	0	∞	−2	∞
2	4	0	**2**	∞
3	∞	∞	0	2
4	∞	−1	∞	0

$k=2$

i \ j	1	2	3	4
1	0	∞	−2	∞
2	4	0	2	∞
3	∞	∞	0	2
4	**3**	−1	**1**	0

$k=3$

i \ j	1	2	3	4
1	0	∞	−2	**0**
2	4	0	2	**4**
3	∞	∞	0	2
4	3	−1	1	0

$k=4$

i \ j	1	2	3	4
1	0	**−1**	−2	0
2	4	0	2	4
3	**5**	**1**	0	2
4	3	−1	1	0

Runtime: $O(n^3)$

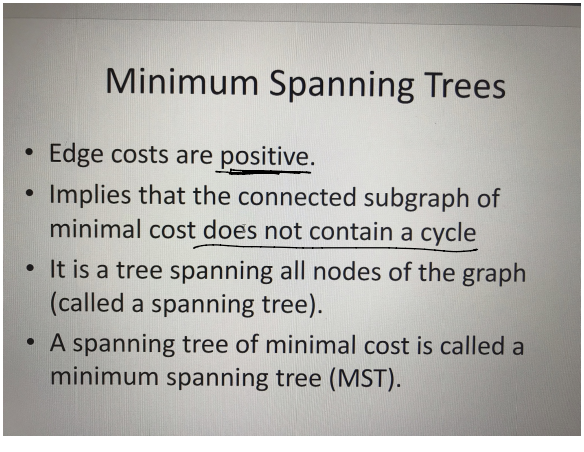
Minimum Spanning Trees

Week 10

最小生成树.

定义: Given a connected undirected graph $G=(V,E)$ with positive edge costs $c: E \to R$
Find a connected subgraph T of G that has minimal cost.

例子: Nodes are the computers of the network
Edges are possible connections between computers
Costs are the cost of establishing a particular connection.



An MST of a given graph G can be constructed by **Greedy Algorithm.**
贪心算法.

Crucial Properties:

cut property: Let e be an edge of minimum cost in a cut C. Then there is an MST that contains e.

cycle property: an edge of maximal cost in any cycle does not need to be considered for computing an MST.

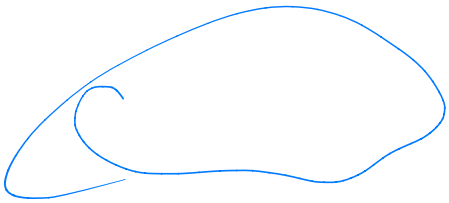
Kruskal's algorithm

Sort in the edge of the graph with respect to increasing weights.
Introduce the edges in ascending offer that do not create cycles.

Runtime: Sorting of the edges takes time $O(m \log m)$

We need to

- test whether an edge connects two different components
- join the two components when an edge is introduced.

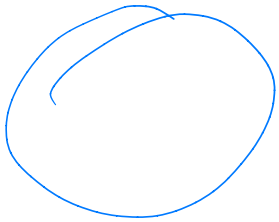


https://youtu.be/tr2NjD-K13Q

Union Find Data Structure 联合查找数据结构.

List-oriented data structure

Jarnik-Prim Algorithm



Prim Algorithm	Kruskal Algorithm
$O(m + n \log n)$	$O(m \log m)$
more efficient for dense graphs. i.e. where $m = \Theta(n^2)$ holds	

P & NP

week 11

Examples

Problems that can be solved in polynomial time:

- Integer Addition - O(n)
- Integer Multiplication – $O(n^2)$
- Test whether a graph is acyclic
- Shortest paths – Dijkstra $O(m+n^2)$
- Minimal spanning trees –Kruskal O(m log m).
- Almost all problems that we consider in this course.

Difficult Problems

There are many problems for which no efficient algorithm is known.

Examples (see Mehlhorn/Sanders page 54):

- Hamiltonian cycle problem
- Traveling Salesman Problem
- Boolean Satisfiability Problem
- Clique Problem
- Graph Coloring Problem

Class NP
Guess an answer
Verify the answer true/false
If we can use polynomial time to verify the answer is true.
The the problem is NP

Class NP

- How can we prove a problem is NP?
  - A problem is not in NP if its solution cannot be verified in polynomial time.
- Any example?
  - All optimisation problems, whose answers cannot be checked in polynomial time.
  - TSP
  - Bin Packing
  - Timetabling

Data Structures and Algorithms

NP-Complete Problems

- We don't know whether polynomial time algorithms exists for the mentioned problems.
- It is very likely (and almost all people in computer science believe) that there are no polynomial time algorithms for these problems.
- They belong to a class of equivalent problems known as NP-complete problems. (NP stands for "nondeterministic polynomial time")

Data Structures and Algorithms

Class **P**

- A decision problem is polynomial solvable iff its characteristic function is polynomial-time computable.
- We use **P** to denote the class of polynomial-time-solvable decision problems.

能在 polynomial 时间内解决的问题

MST problem 是 Class P ?

—Hint : Upper bound on Kruskal

$O(m \log m) < O(m^2)$