CS6033 Lectures 11-12 Slides/Notes

Greedy Algorithms; Minimum Spanning Trees (Notes, Ch 15, Ch 21)

By Prof. Yi-Jen Chiang
CSE Dept., Tandon School of Engineering
New York University

1

Greedy Algorithms (ch 15)

Typically for optimization problems. Simpler than problems that require D.P.

Greedy Algorithms can be viewed as an improvement over D.P. when applicable.

Ex: Activity Selection Problem: Given one machine and n jobs.

job i: start time s_i, finish time f_i.

jobs i, j are compatible if

i j: s_i f_i s_j f_j, $f_i \le s_j$.

or vice versa.

j i: s_j f_j s_i f_i, $f_j \le s_i$

e.g.

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

$\boxed{1, 4, 8, 11}$

O(n) time after sorting jobs into increasing finish time.

2

2 major properties for Greedy Algorithms.

1. Greedy-choice Property

   There is an optimal sol. with the (first) greedy choice.

   Typically we use a swapping argument in the proof.

2. Optimal Substructure (the same as in D.P)

   The optimal sol. for the current problem contains an optimal sol for the subproblem. (For greedy alg. we have 1 subproblem)

   Optimal sol. with greedy choice contains an opt. sol. for the subproblem.

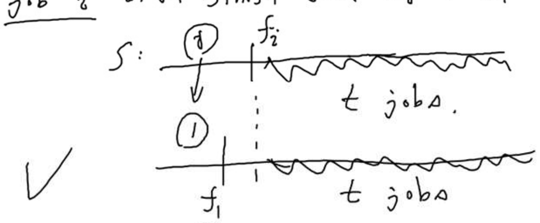
   We use "cut & paste" argument in the proof.

3

pf for the greedy alg. for activity selection:

(1) greedy-choice property.

Suppose S is an optimal sol that starts with some job i with finish time $f_i > f_1$



opt. sol S: $t+1$ jobs.

$t+1$ jobs. optimal

(2) Optimal substructure.

Let S' be an optimal sol. starting with job 1 (i.e. with greedy choice)

S': ① $|$ f_1 $\underbrace{\boxed{t \text{ jobs.}}}_{S_1'}$ $\rightarrow$ If $\exists S_2' \geq t+1$ jobs $\Rightarrow$ $\underbrace{\{1\} \cup S_2'}_{\geq t+2 \text{ jobs}}$ $\Rightarrow$ S' is NOT optimal

4

Knapsack Problem (Textbook P430)

A set of items 1, 2, 3, ... i ..., n

weight w_i

benefit b_i

Knapsack: capacity W.

Goal: put items to the knapsack s.t. total weight $\le W$ and the total benefit is max.

Eg. item 1 item 2 item 3

$w_1 = \boxed{10}$ $w_2 = \boxed{20}$ $w_3 = \boxed{30}$

\$60, \$100, \$120.

$W = 50$.

0-1 Version:

\$100 $\boxed{20}$, \$60 $\boxed{10}$ Total: \$160

$\boxed{30}$ \$120, $\boxed{10}$ \$60 \$180.

$\boxed{30}$ \$120, $\boxed{20}$ \$100 \$220.

DP. Greedy-choice property can NOT be satisfied

Fractional Version

Greedy: $\frac{\text{benefit}}{\text{weight}}$.

Item 1: $\frac{60}{10} = 6$ ✓

Item 2: $\frac{100}{20} = 5$ ✓

Item 3: $\frac{120}{30} = 4$

$120 \cdot \frac{2}{3} = 80$

\$80 $\boxed{\frac{20}{30}}$

\$100 $\boxed{20}$

\$60 $\boxed{10}$

Total: \$240

5

Huffman Coding: Data Compression.

Encode symbols into binary-bit code words 01001...

Idea: Give symbols appearing more frequently: shorter code words

" " less " : longer " "

so that Total file length is minimized.

Ex:	a	b	c	d	e	f
frequency: (in k)	45	13	12	16	9	5.
	000	001	010	011	100	101
	0	101	100	111	1101	1100.

fixed-length code: 3 bits per symbol.

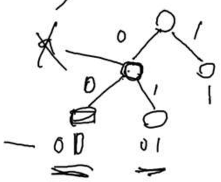
File length $= 3 \cdot 45 + 3 \cdot 13 + 3 \cdot 12 + \cdots$
$= 3(45 + 13 + 12 + \cdots)$

Total length: $1 \cdot 45 + 3 \cdot 13 + 3 \cdot 12 + 3 \cdot 16 + 4 \cdot 9 + 4 \cdot 5$.

6

* Prefix code : (prefix-free) code

Binary bits : Binary tree to represent the codes



symbol	
a	0 #
b	0 0 #

If (a is a prefix of b)

0# 0#0#

0 | 0 | 0
a | a | a

0 | 0 0
a | b

0# 00#

0 0 | 0
b | a

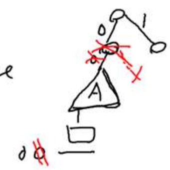
00# 0#
b a

Need a special "#".
⇒ Extra length,
No Good !

So we want

Prefix code ⇒ Every code is at a leaf.

Optimal code : Each internal node of the binary tree has 2 children (full binary tree)

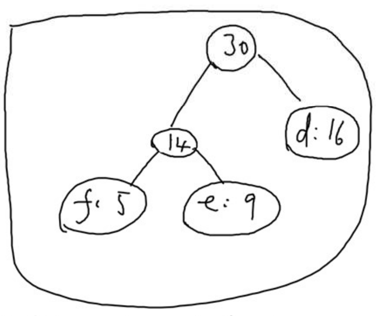


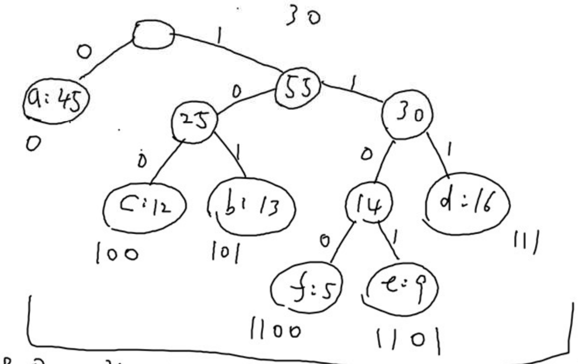
7

a	b	c	d	e	f
45	13	12	16	9	5

25 14

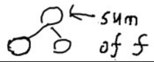
30





* Put each symbol to the P. Q. with frequency as the key.

* In each iteration, Perform [Extract_Min( ) twice. Insert the parent with new f] $O(\log n)$ time



\# iterations: $n \xrightarrow{(n-1)} 1$

$\boxed{\text{Total time } O(n \log n)}$

8

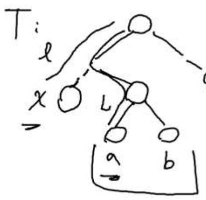
Greedy-choice Property

Lemma: There is an optimal binary prefix code with symbols x, y having the max code length where x, y have the smallest frequencies.

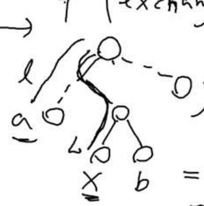
Pf: Let T be a binary tree representing an optimal code. where a, b are the 2 symbols with the max code length.

$\underbrace{x.f < y.f}_{\text{smallest frequencies}}$ $\quad a.f < b.f$. $\quad a, b, x, y$ are distinct.

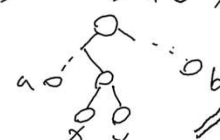
T:



$\longrightarrow$ T' (exchange x & a)



$\longrightarrow$ T'' (exchange y & b)



$$B(T') - B(T) = x.f \cdot L + a.f \cdot l - (a.f \cdot L + x.f \cdot l) = L(x.f - a.f) + l(a.f - x.f)$$

$$\underbrace{(a.f - x.f)}_{\geq 0}\underbrace{(l - L)}_{<0} \leq 0$$

i.e. $\boxed{B(T') \leq B(T)}$

$\boxed{\text{similarly, } B(T'') \leq B(T')}$

$\Rightarrow B(T'') \leq B(T)$ #

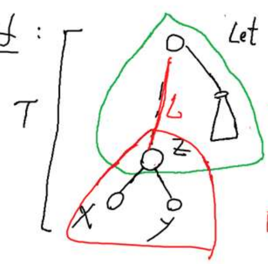
9

Optimal substructure

Lemma: Let x, y be 2 symbols of the least frequencies. Let z be a symbol with $z.f = x.f + y.f$. Then the optimal code for n symbols (including x, y) contains an optimal code for $n-1$ symbols (excluding x, y & including z)

Pf: Let $B(T)$ be optimal for n symbols

T



T': $z.f \cdot L$

T: $x.f \cdot (L+1) + y.f \cdot (L+1)$

$$B(T) - B(T') = \underbrace{(L+1)\,(x.f + y.f)}_{} - z.f \cdot L \qquad (z.f = x.f + y.f)$$

$$= (x.f + y.f) \cdot L + (x.f + x.f) \cdot 1 - (x.f + y.f) \cdot L = x.f + y.f$$

$$B(T) = (x.f + y.f) + B(T') \qquad B(T'')$$

If $B(T')$ is not optimal for $n-1$ symbols, say $B(T'')$ is better then $B(T'') + (x.f + y.f)$ is better than $B(T)$.

10

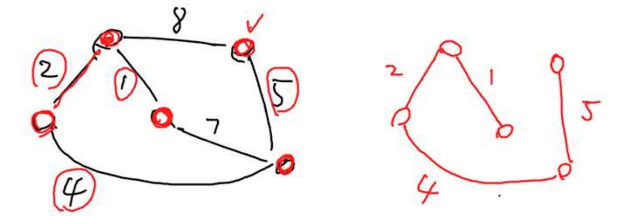
Minimum Spanning Tree (MST) Problem : (Ch 21)

Given an undirected, weighted, connected graph $G=(V,E)$.

Find a spanning tree (a tree that connects all vertices of V)

s.t. the total edge weight in the spanning tree is minimum possible.

Eg:



11

Generic Greedy Alg. (Sec. 21.1) $G=(V,E)$.

* Let A be a subset of edges that belong to some MST

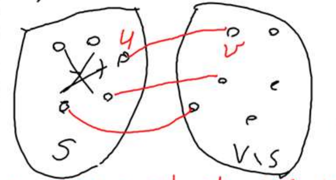
Def: A safe edge e of A is an edge s.t.

$\{e\} \cup A \subseteq$ some MST.

Alg: In each iteration.
- find a safe edge e for A.
- $A \leftarrow A \cup \{e\}$.

Repeat until A spans all vertices (A is a spanning tree).

* A cut $(S, V\setminus S)$ is a partition of vertex set V into 2 subsets S, $V\setminus S$.

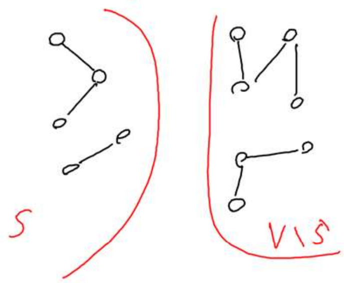


(u,v) is a cut edge if $u \in S$ & $v \in V\setminus S$.

* Light edge of a cut: The cut edge with min length among the cut edges

12

* A cut respects the edge set A:

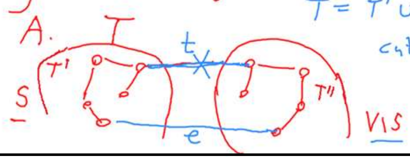


— : edges of A.

No edge of A goes across the cut.

$\boxed{\text{Thm 21.1 in textbook}}$: The light edge of a cut respecting edge set A is a safe edge for A.

Pf: Let T be an MST containing all edges of A. e is a light edge of a cut $(S, V \setminus S)$ respecting A.



$T = T' \cup T'' \cup \{t\}$

cut respects $A \Rightarrow t \notin A$.

$\ell(e) \le \ell(t)$

$\tilde{T} = T' \cup T'' \cup \{e\}$ is MST

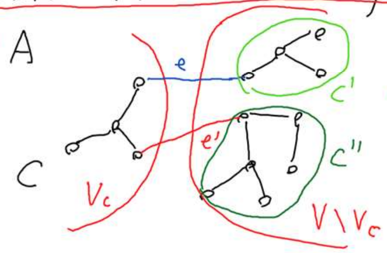
$\tilde{T} \le T$.

13

Corollary: $G = (V, E)$. Let A be an edge set cotained in some MST.

Let C be a connected component of A (C is a tree in the forest of A). The light edge connecting C to some other connected component of A is a safe edge for A.

Pf: A



V_c : vertices in component C.

Cut : $(V_c, V \setminus V_c)$ respects A.

if e is a cut edge and $e \in A$ then C and C' would be the same c.c. $\rightarrow\!\!\!\times$

$\Rightarrow$ Any cut edge $e \notin A \Rightarrow$ the cut respects A.

Now: Suppose e' is a light edge of the cut $(V_c, V \setminus V_c)$.

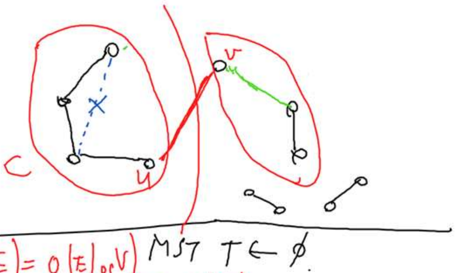
Then e' is a light edge of a cut respecting A. $\Rightarrow e'$ is a safe edge for A.

14

Two main algs. for MST

1. Kruskal's Alg.

2. Prim's Alg.



1. Kruskal's Alg.: $G=(V,E)$ $\rightarrow O(E\log E) = O(E\log V)$ MST $T \leftarrow \phi$. $O(E)$

① Sort all edges from shortest to longest.

② Consider edges in the sorted order.   Union - Find data structure

$\boxed{\text{Total time: } O(E \log V)}$

For the current edge (u, v):   Find-set(u)   Find-set(v)

If u, v are in different connected components

then $T \leftarrow T \cup \{(u,v)\}$ union the c.c of u

Union (u, v); $\longrightarrow$ with c.c. of v   V items.

else // u, v are in the same c.c.

ignore (u, v)

$O(E)$ union-find ops.

$O(E + V\log V)$ time. (linked lists)

$O(E\,\alpha(V))$. time (advanced)

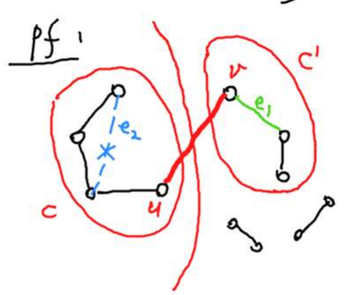
15

Correctness of Kruskal's Alg:

Let A be the set of edges of T so far

Claim: The current edge (u,v) added to T is a safe edge for A

Pf:



The only edges shorter than (u,v) are those considered before (u,v).

2 types: ① those added to A, eg. e_1

② those ignored. eg. e_2

$\Rightarrow$(a) They never connect 2 connected components of A

$\Rightarrow$(b) (u,v) is the shortest edge that connects 2 c.c.s of A

By Corollary, (u,v) is a safe edge for A ✱

Another way,

(a) They never go across the cut

(b) (u,v) is the shortest edge across the cut i.e., light edge of the cut

16

Disjoint-Set Union-Find Data Structure

n items. Initially each item is in a single set.

2 ops:

1. Find_set(x): Find the set containing item x.
2. Union(x, y): $A \leftarrow$ Find_set(x) If $A \neq B$, $B \leftarrow$ Find_set(y) union the sets A and B.

Initially, n sets
At the end: at least 1 set

Each union reduces # sets by 1.
$\Rightarrow$ At most $n-1$ union ops.

A sequence of m intermixed Find-set & Union ops.

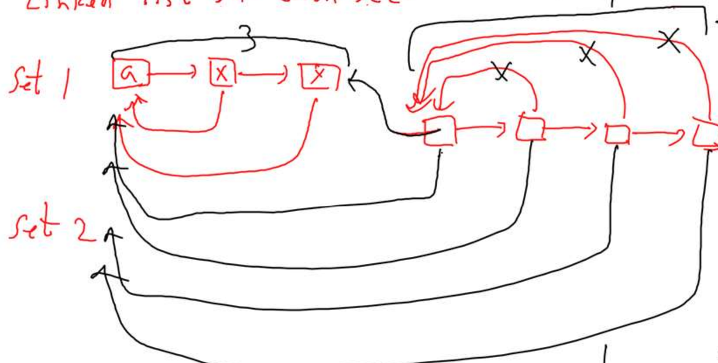
$\Rightarrow$ $O(m\,\alpha(n))$ time

$\alpha(n)$: inverse Ackermann's function
$\alpha(n) \le 4$ for $n \le \underbrace{10^{80}}$ estimated # particles in the universe.

17

Simpler method for union-find

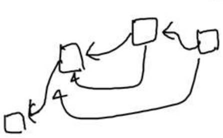
Linked list for each set.



Time for union:
Length of sencond set

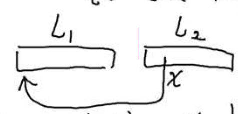
Find: $O(1)$

Better method:
Append shorter set to the longer set.



1
2
3

$1+2+3+\cdots+(n-1) = O(n^2)$



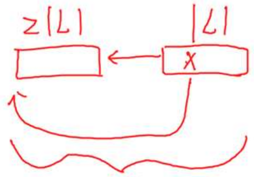
$|L_2| \le |L_1|$

For each item x, how many times can we update the ptr of x?

18

Key pt: Each time we update the ptr of an item x, x is in a set whose size is at least doubled

$2|L|$ $|L|$



size of the set containing x:
1
2
4
⋮
$2^k \le n$.

$\ge 2|L|$ $\Rightarrow k \le \log_2 n$.

$\Rightarrow$ Total update time for all union ops: $O(n \log n)$

Find: $O(1)$ time m ops

Total time for union-find sequence: $O(m + n\log n)$ ✓

v.s.

$O(m\,\alpha(n))$

19

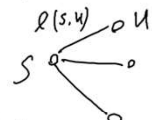
---

2. Prim's Alg. Always grow the same connected component.

$T \leftarrow \emptyset$

① Pick an arbitrary vertex s as the component to grow.

② Put all other vertices into a P.Q Q with initial key value ∞. except for neighbors of s.:



$key(u) \leftarrow l(s,u)$

$key\text{-}edge(u) \leftarrow s$.

Each neighbor u of s has $key(u) \leftarrow l(s,u)$.

③ In each iteration. #: V

$u \leftarrow$ Extract_Min(Q)

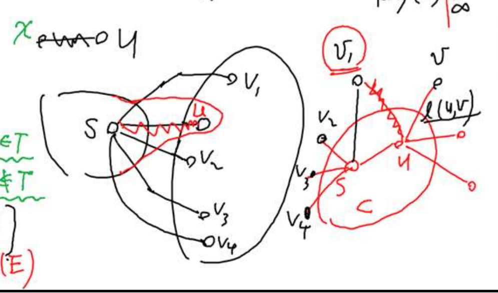
Let x be the key_edge(u)

$T \leftarrow T \cup \{(u,x)\}$. mark u to indicate $u \in T$

for each neighbor v of u s.t. $v \notin T$

$key(v) \leftarrow \min\{key(v), l(u,v)\}$

DecreaseKey$(v, l(u,v))$ #: $O(E)$



Implicit Binary Heap: $O(\log V)$ time per Extract_Min, Decrease_key

$\Rightarrow O((V+E)\log V)$ time

Fibonacci Heap: $\Rightarrow O(E + V\log V)$ time

$key(v) = \infty$

$Key(v_1) = l(s,v_1)$

$Key(v_1) \leftarrow \min\{Key(v_1), l(u,v_1)\}$

20

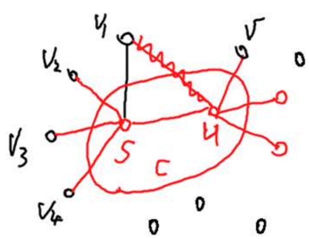
Correctness of Prim's Alg:

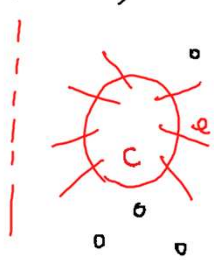
Let A be the set of edges of T so far

Claim: The next edge added to T is a safe edge for A

Pf: Let C be the set of vertices spanned by T so far

Look at the cut $(C, V \setminus C)$





The next edge e added to T is the shortest edge among the edges that connect from inside C to outside C

i.e. e is the shortest among the cut edges

$\Rightarrow$ e is a light edge of the cut

thus a safe edge for A ※

21

More Details: Decrease_key(v, k, Q) in Priority Queue Q

③ In each iteration. → #: V

$u \leftarrow$ Extract_Min(Q)

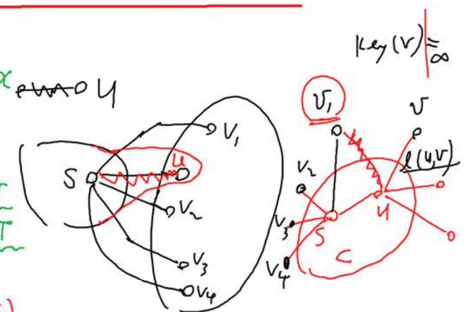
Let x be the key_edge(u)

$T \leftarrow T \cup \{(u, x)\}$. mark u to indicate $u \in T$

for each neighbor v of u s.t. $v \notin T$

key$(v) \leftarrow \min\{$key(v), $\ell(u,v)\}$

DecreaseKey$(v, \ell(u,v))$ #: $O(E)$

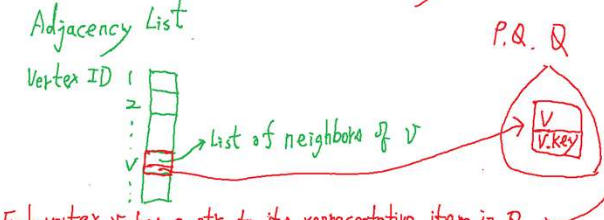


Key$(v_1) = \ell(s, v_1)$

Key(v_1) $\leftarrow \min\{$Key(v_1), $\ell(u, v_1)\}$

* How do we access v in Q efficiently?

Sol: Adjacency List



→ such ptr supports access to v in $O(1)$ time

in general, this is needed for Decrease_key(v, k, Q) & Delete(v, Q) for Q

\+ Each vertex v has a ptr to its representative item in Q.

\+ In Adjacency List, we can also mark u for "$u \in T$".

22