Topological Ordering

9 | 20 | 22

Every DAG G, has a node with no incoming edge

Proof by contradiction
- Pick arbitrary node
- Visit every

Proof by induction
- Removing a node implies a DAG still exists

Greedy Algorithm

decision based on state

Locally optimal | Heuristic / calculation

→ for given problem, many Greedy Algorithms can exist

i.e. Priority Queue
We know entire input (offline)
The algorithm process input in defined order
→ The last input has the least greedy cost to process (Performance)

Bin Packing Problem

Bins of size 1 ← requests size $(0, 1]$

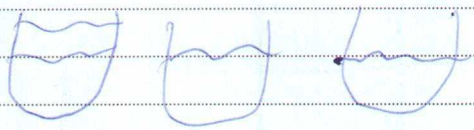
Objective:
Pack items in minimum # of bins

Greedy Heuristic: First Fit Increasing (FFI)

Non optimal $\delta = (\frac{1}{2}-\varepsilon, \frac{1}{2}-\varepsilon, \frac{1}{2}+\varepsilon, \frac{1}{2}+\varepsilon)$

FFS: 3 Bins

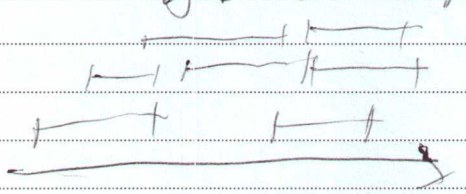
Optimal fit:
2 bins



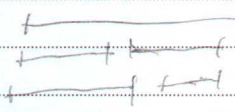
Show Greedy is optimal
- Always Stays ahead (proof usually by induction)
- ← Exchange Argument

Stay Ahead: Interval Scheduling

- Request $\mathcal{R} = \{r_1, \ldots, r_n\}$
- request $r_i = (s_i, f_i)$ s_i is start time f_i is finish time
- Objective: Compatible schedule S, $\forall r_i, r_j \in S, f_i \leq s_j \vee f_j \leq s_i$



```
Let S be initially empty
While R is not NULL
  choose r_i ∈ R w/ smallest finish time
  Add r_i to S
  Remove all incompatible
end
return S
```



TÜL

Stay Ahead Analysis

Lemma: $\forall i_r, j_r \; r \le k \rightarrow f_{i_r} \le f_{j_r}$

Proof: For $r=1$
The claim holds FF

Assume $r-1$
— induction hypothesis $f_{i_{r-1}} \le f_{j_{r-1}}$
— The only S is behind S^* is

Algorithm First Finish
Let S be initial empty set

While $\mathcal{R}$ is not Empty do
choose $r_i \in \mathcal{R}$ with the smallest finish time
(Ties goes arbitrarily)
- Add r_i to S
- Remove all incompatible requests in $\mathcal{R}$
End
Return S

- choose request with smallest finish time: Before Processing Sort request: $O(n \log n)$
- Remove incompatible requests advance in sorted order until a request with compatible start time

Overall
$O(n \log n) + O(n) = O(n \log n)$

---

n jobs and a single machine that can process one job
For Job j: t_j is processing time, d_j is due
- Lateness $l_j = f_j - d_j$ if finish time $f_j > d_j$; 0 otherwise

Write Scheduler

J_1 | deadline 2
J_2 | deadline 4
J_3 | deadline 6

Heuristic 1 — Increasing processing time
- schedule jobs by increasing t_j

not optimal Jobs (t_j, d_j): $\{(1, 100), (10, 10)\}$

1.) Define Solution
S = greedy
M = optimal
2.) Compare
$S \neq M$
3.) Exchange
4.) Iterate - while $(S \neq M)$ or till $S = M$

Heuristic Increasing slack
Not optimal
counter example
Jobs (t_j, d_j): $\{(1,2), (10,10)\}$
OPT
1-10 = -1
1-2 = -1
latenes

Heuristic schedule by increasing d_i

algo: EDF
Let J be set of jobs
Let S be an initially empty list
While J not empty do
  choose $j \in J$ with smallest d_j (break ties)
  Append j to S
End
Return S

Observation 2: There is an optimal schedule with no idle time

Showing Optimal — Let S^* be an optimal solution

- Is it sufficient to show that $|S| = |S^*|$? NO
- Can there be multiple S^*? yes
- We need to show either $S = S^*$ or $S \equiv S^*$ for max lateness

Technique: — Start with an optimal solution S^* and transform it over a series of steps to something equivalent to S while maintaining optimality.

$S^* = S_1 \equiv S_2 \equiv \dots \equiv S$ for max lateness

Definition 3: a schedule A has an inversion if there are jobs i and j with i scheduled before j and $f_i < d_j$

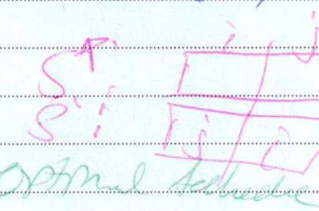
Lemma 4
All schedule with no inversions and no idle time have the same lateness.

Proof:
- only swapping jobs w/ same lateness
- jobs w/ same deadline must be sequential
- ordering of jobs with same deadline won't change lateness

Theorem 5: There is an optimal schedule that has no inversions and no idle time

Proof:
- If S^* has an inversion, then there is a pair of jobs i and j s.t. j is scheduled immediately after i and has $d_j < d_i$
- we swap i and j to create new schedule S'. note S' has one less inversion than S^*
- we need to show that S' has the same max lateness as S^*
  - Swapping i & j means that l_j' (lateness S') less than l_j in S^*
  - lateness in i may increase but:

$$l_i' = f_i' - d_i = f_j^* - d_i \leq f_j^* - d_j = l_j^*$$



Corollary 6: EDF Produces optimal schedule

Proof:
- EDF produces a schedule w/ no inversions and no idle time
- from Theorem 5 there is an optimal schedule w/ no inversions and no idle time

Shortest Path

Problem Def: Assume we have Directed graph, graph $G = (V, E)$ where $|V| = n$ and $|E| = m$ and a node s that has a path to every other node in V.

For each edge e $l_e \geq 0$ is the length of the edge

- What is the shortest path.

Kruskal (1956)
- Short edge by cost from lowest to highest
- insert edge unless insertion would create cycle

Thm 11 → Kruskals produces MST

Proof

Prim's (1957)

Thm 12 prim's produces MST

Proof
- Immediate from lemma 10
- That is, Prim's algorithm does exactly what lemma 10 describes

Reverse-delete

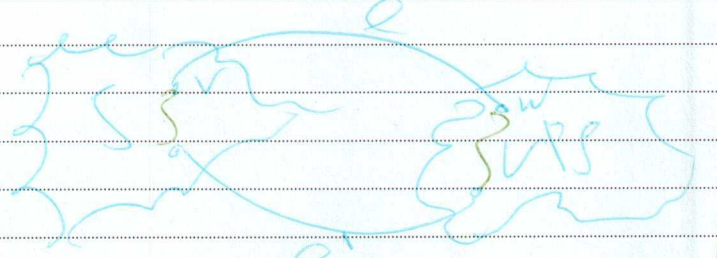
Thm

~~Proof~~ proof

Let $e = (v,w)$ be an edge at any step i

be def e belongs to cycle C

as c_e is the maximum



Lemma 13

Let C be any cycle in G and let e be the most expensive edge of C.

Then e is not in any MST of G

Proof
- Let T be a spanning tree w/o e
- Let S and V/S be vertices of connected components after removing e from T
- Let $T' = T \cup e'$
- T is connected as e' reconnects
- Since $c_{e'} < c_e$ cost of T' is ~~less~~ less than T.

Prim's

Union-find data structure

Find(x): finds the set containing x (O(log n) can be O(α(n)). ← inverse ackermann function, almost constant

union(x, y): joins two sets x and y: (O(1)) given

node	rank	parent

initialize

Find(x)

If x, ⇒ parent points to x

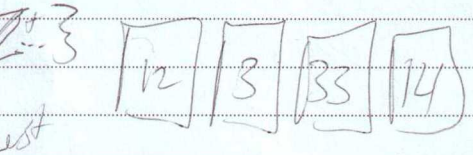
Else Find(x.parent)

O(log n) // requires balanced trees

O(α(n)) // with path compression

Paging Problem

- U: universe of pages (|U| > k)
- Cache of size k
- Requests are to the page of U
- Goal: Minimize the number of page faults (requests to pages not in cache)



12, 3, 33, 14, 12

Eviction Strategies

- When designing an algorithm, we are picking an eviction strategy
- In the offline version, the algorithm knows the request sequence. (least recently used, lazy algorithm ↳ do swap when required to do so)

Algo: Dijkstra

Let S be the set of explored nodes
For each $u \in S$, we store a distance value $d(u)$
Initialize $S = \{s\}$ and $d(s) = 0$

while $S \neq V$ do
  Choose $v \in S$ with at least one incoming edge originating from a node in S with smallest
  $$d'(v) = \min_{e=(u,v): u \in S} \{ d(u) + l_e \}$$
  Append v to S and define $d(v) = d'(v)$
end

Theorem 7 Dijkstra's Proof
For $|S| = 1$ trivial $S = \{s\}$

By induction
- In step $k+1$ we add v
  By definition P_v is shortest path connected to S by one edge
- Since P_u is a shortest path to u P_v is the shortest path to v wrt nodes $\in S$

- Also, there cannot be a shorter path to v passing through another node $y \notin S$ else y would be added $k+1$

Minimum Spanning Tree

Let $G = (V, E)$ be connected graph,

$|V| = n$ and $|E| = m$

For each e, $c_e > 0$ the cost of the edge.

- Find set $F \subseteq E$ with minimum cost

$$\sum_{e \in F} c_e$$

Proof

- By definition of the problem, T must be connected
- By contradiction, assume T has a cycle C

Remove any edge from C resulting in graph T'.
T' is still connect and has a cost less than T

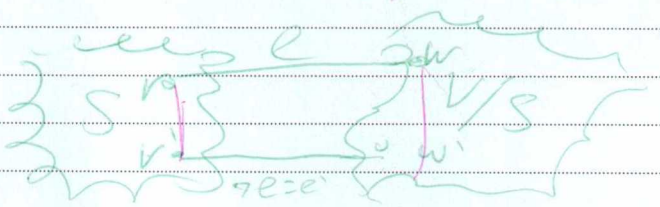
If all edges weight are distinct then G has unique MST

Cut properties.

Let $S \subset V$ be non-empty proper subset of the nodes and let $e = (v, w)$ be the minimum cost edge connecting S and $V \setminus S$. $\rightarrow$ $\forall$ MST has e

Proof by exchange argument

- Let T be a spanning tree that doesn't have e
- Let $e' = (v', w')$ where e' is in $P_{v,w}$, $e' \in T$, $v' \in S$ and $w' \in V \setminus S$
- Let $T' = T \setminus e' \cup e$
- Since $c_e < c_{e'}$ cost of T' is less than T

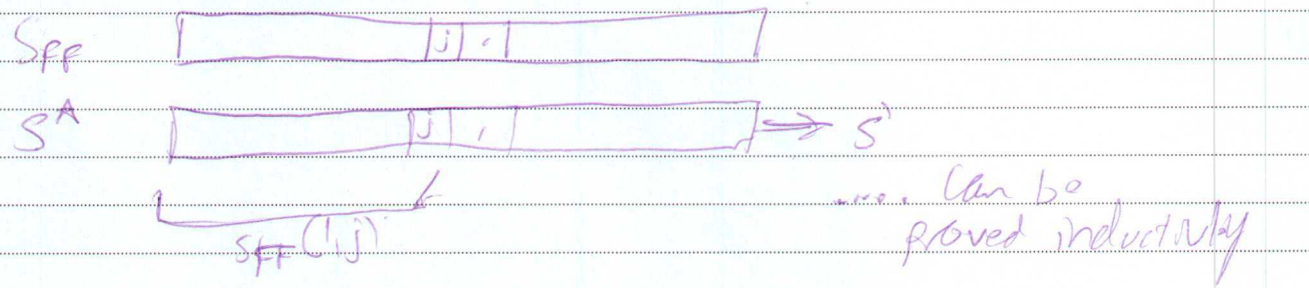


Farthest in the future (FF)

Evicts the pages whose next request is the furthest into the future

Theorem 8

Let S be a schedule for the n request that makes the same eviction decisions as S_{FF} for the first j items. Then there is a schedule S' that makes the same eviction requests as S_{FF} for the first $j+1$ items with no more faults than S.



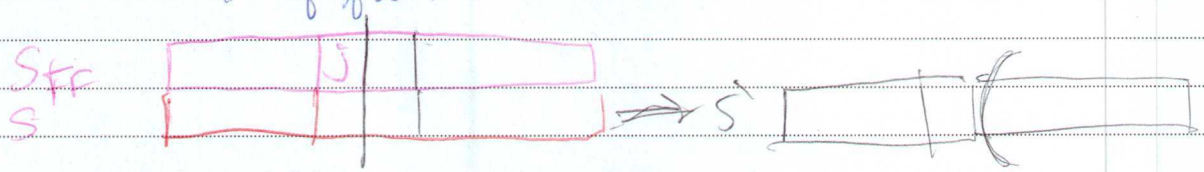
..... Can be proved inductively

Proof

- If on request $j+1$, S behaves as S_{FF}. Then define S' as S, chain follows

~~new chain~~

- Else Suppose S evicts u and S_{FF} evicts v. We build S' by following S_{FF} for the first $j+1$ requests. Note that the # of faults



- from $j+2$ onward, S' follows S until either
  - S evicts v.
    - S' evicts u
  - S evicts some page $g \neq v$ to bring u in to cache k
    - S' evicts g and brings in v

Note: Since S_{FF} evicts v at $j+1$, u must be requested before v

- In either case, both S' & S have a page fault and have the same cache k

K-Clustering

Maximum Spacing Problem

- A universe $U = \{p_1, \ldots, p_n\}$ of n objects
- Distance function $d: U \times U \to \mathbb{R}$ s.t. $\forall p_i, p_j \in U$:
  - $d(p_i, p_i) = 0$
  - $d(p_i, p_j) > 0$
  - $d(p_i, p_j) = d(p_j, p_i)$

Maximum $\min_{C_i, C_j} \min$

Algorithm
- Build an MST
- Remove $k-1$ largest edges

- Start with tree, remove $k-1$ edges
- By definition largest edges are removed (max spacing)

Kruskal's ($O(m \log n)$) where $O(n^2 \log n)$ for clustering
→ Merge sets from lowest to most expensive edges
→ Stop when we have k sets

Binary Encoding

- Set symbol $S = \{a, b, c, d, e\}$
- Encoding function $\gamma: S \to \{0,1\}^k$

Fixed width

Variable width

Divide and Conquer (DC)

10 | 4 | 22

- Split problem into subproblem
- Solve / usually recurse
- use small outputs to build solution

Tendencies of DC
- Natural recursive solution
- solving complexity
- often DC → $O(n \log n)$

Linear Search
- Brute force $O(n)$

DC
- Binary Search $O(\log n)$

Sorting $O(n \cdot n!)$
Insertion Sort, Selection Sort, Bubble sort $O(n^2)$

DC Quick Sort $O(n^2)$, Merge Sort $O(n \log n)$
RadixSort ($O(n \lceil \log K \rceil)$), Counting Sort ($O(n+K)$)
↳ K: is max key size

Merge Sort
Input: A list A of n comparable items
Output: A sorted list A
If $|A| \geq 1$ Then return A
A_1 = MergeSort(Front-half A)
A_2 = MergeSort(Back-half A)
Return Merge(A_1, A_2)

Program Correctness
Invariant: list A is sorted after call to mergeSort.
Proof: By strong induction on list length
Base Case: $k=1$: list is sorted; $k=2$: split into size 1, then merge produces sorted list
Inductive Step: By inductive hypothesis A_1 and A_2 are sorted, and by definition, merge will produce sorted list.

mergeSort call,
① Invariant: list A is sorted after
② Soundness: Immediate from invariant
③ Complete: handles list of any size; each case gets smaller until Base

Merge Sort Recurrence
$T(n) \leq 2 \cdot T(\frac{n}{2}) + cn$; $T(1) \leq$

Constant Base case

$T(n) \leq 2 \cdot T\left(\frac{n}{2}\right) + cn \quad ; \quad T(1) \leq c$

- cost to merge $O(n)$
- Recurrences: 2 calls to mergeSort with left half the size

- More precise $T(n) \leq T\left(\lfloor \frac{n}{2} \rfloor\right) + T\left(\lceil \frac{n}{2} \rceil\right) + cn$
  - asymptotically ignore floor/ceiling
  - assume n is a power of 2
- Aff form: $T(n) \leq 2 \cdot T\left(\frac{n}{2}\right) + O(n)$; $T(1) \leq O(1)$

Method 1: Unwind

$$T(n) \leq 2T\left(\frac{n}{2}\right) + cn$$
$$\leq 2\left(2T\left(\frac{n}{4}\right) + \frac{cn}{2}\right) + cn$$
$$\leq 2\left(2\left(2T\left(\frac{n}{8}\right) + \frac{cn}{4}\right) + \frac{cn}{2}\right) + cn$$
$$\vdots$$
$$\leq 2^k T\left(\frac{n}{2^k}\right) + kcn$$

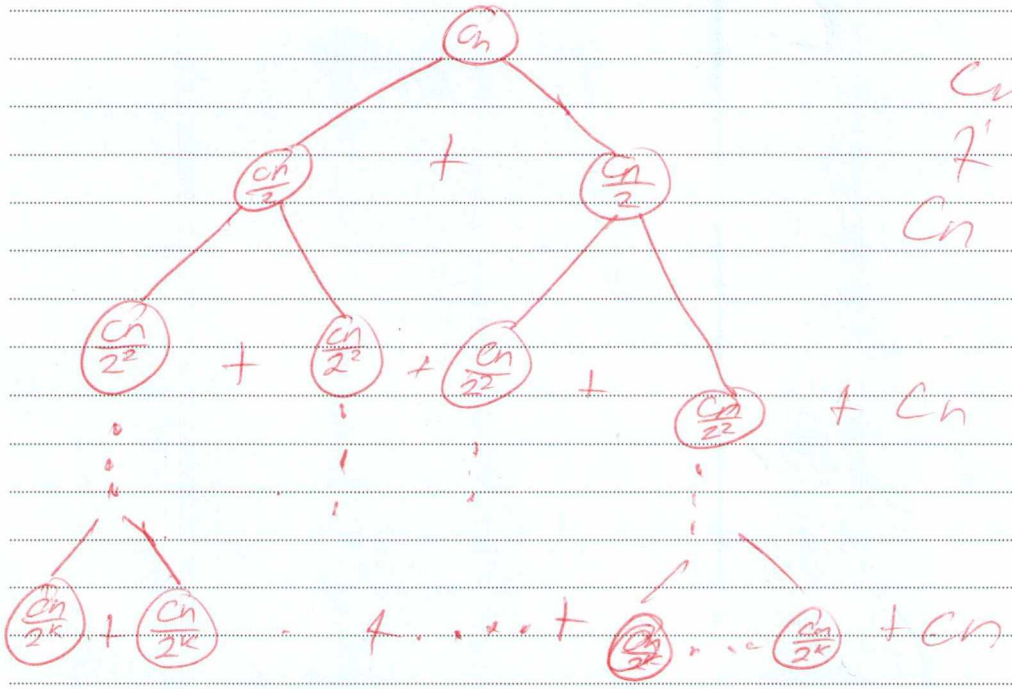
$1 = \frac{n}{2^k} \iff 2^k = n \iff k = \log_2(n)$

$2^k = n$

$$\vdots$$
$$n \cdot T(1) + cn \cdot \log_2(n)$$
$$cn + cn \log n$$
$$= O(n \log(n))$$

Recursion Tree Method:

$T(n) \leq 2 \cdot T(\frac{n}{2}) + cn$; $T(1) \leq c$



$$\left(\sum_{i=0}^{k} 2^i \cdot \frac{cn}{2^i}\right) = (k+1)cn = (\text{tree height} + 1) \cdot cn$$

$$= cn \cdot \log n + cn$$

$$= O(n \cdot \log n)$$

Guess Method

$T(n) \leq 2 \cdot T(\frac{n}{2}) + cn$, $T(1) \leq c$

- make an educated estimation
- prove by Induction

Induction Step

$T(k) = 2 \cdot T(\frac{k}{2}) + ck \leq 2\left(\frac{ck}{2}\log\frac{k}{2} + \frac{ck}{2}\right) + ck$

$= ck \log(k/2) + 2ck$

$= ck \log k - ck + 2ck$

$= ck \log k + ck$

Show $T(n) \leq 2 \cdot T(\frac{n}{2}) + cn \leq cn \log n + cn$, $T(1) \leq c$

Base Case

$n = 2$

$T(2) = 2 \cdot T(1) + 2c \leq 4c$

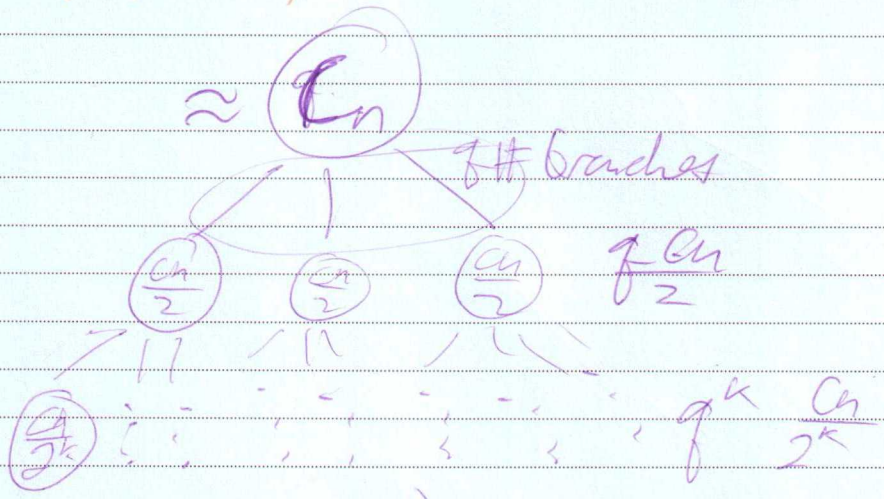
$= c \cdot 2 \log_2 2 + 2c$

Generalized Recurrence

$T(n) \leq q \cdot T\left(\frac{n}{2}\right) + cn; \; T(1) \leq c$

Case $q > 2$

$O(n^{\log q})$



$$T(n) = \sum_{i=0}^{k} q^i \frac{cn}{2^i} = \sum_{i=0}^{k} cn \left(\frac{q^i}{2^i}\right)$$

let $r = \frac{q}{2}$ $\quad i = k$

$$= cn\left(\frac{r^{k+1} - 1}{r - 2}\right)$$

$$q^k \cdot \frac{cn}{2^k} \leq cn\left(\frac{r \log n + 2}{r - 1}\right)$$

$$= O\left(n \cdot n^{\log\left(\frac{q}{2}\right)}\right)$$

Inversion ~~Counting~~

Given a list A of comparable items. An inversion is a pair of items (a_i, a_j) $a_i > a_j$ on $i < j$ where i & j are the index in A

Inversion Counting

- Count the number of inversions in a list A containing n comparable items.

Algorithm: Check All Pair

Input: a list A of n comparable items
Output: Number of inversions A

```
Let C := 0
for i := 0 to len(A) - 1
    for j := i + 1 to len(A)
        if A_i > A_j
            C = C + 1
    End
end
return C
```

- correct check all pairs counts all inversions
- complexity

$$\sum_{i=1}^{n} \frac{n(n-1)}{2} = \Theta(n^2)$$

Algorithm: count sort

Input: a list A of n comparable items
Output: a sorted array and the number of inversions

$O(n \log n)$

```
if |A| = 1 then return (A, 0)
(A_1, C_1) = CountSort(front half of A)
(A_2, C_2) = CountSort(Back half of A)
(A, C) = MergeCount(A_1, A_2)
return (A, CF + C_1 + C_2)
```

Linear Time Selection

problem

Find k^{th} value in unsorted array A of n if A were sorted

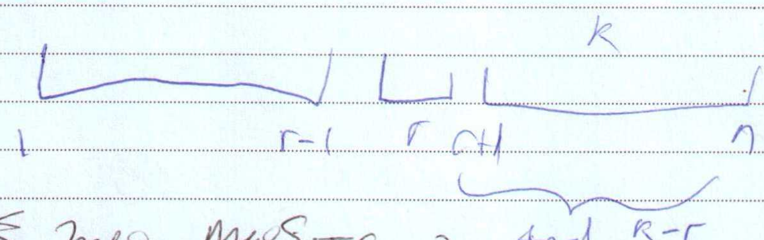
Algorithm QuickSelect

Input: A array $A[1...n]$ and an int k

Output: The k^{th} element of A if A were sorted

```
if n=1 then return A[1]
    choose pivot A[p]
    r := Partition(A[1..n], p)
if k<r then
    return QuickSelect(A[1...r-1], k)
else if k>r then
    return QuickSelect(A[r+1...n], k-r)
else
    return A[r]
end
```

$O(n)$



$$T(n) \le \max_{1 \le r \le n} \max\{T(r-1), T(n-r)\} + cn$$

$$\le \max_{1 \le l \le n-1} T(l) + c_n$$

$$\le T(n-1) + c_n$$

$$\in O(n^2)$$

Median of Medians (MomSelect)

Input
Output

if n is small then solve by brute force
m ← ⌈n/5⌉
for i ← (1 → m) do
  M[i] = brute force median of A[5i−4 ... 5i]
end
mom ← MomSelect(M[1...m], ⌈m/2⌉)
r ← Partition(A[1...n], mom)
if k < r then
  return MomSelect(A[1...r−1], k)
else if k > r then
  return MomSelect(A[r+1...n], k−r)
else
  return A[r]
end

Split groups of 5

MomSelectPivot Analysis

- greater and less than → ⌊⌈n/5⌉/2⌋ − 1 ≈ n/10 medians.
- i.e. momselectPivot is great and less than 3n/10 items
- So worst case partition size is 7n/10

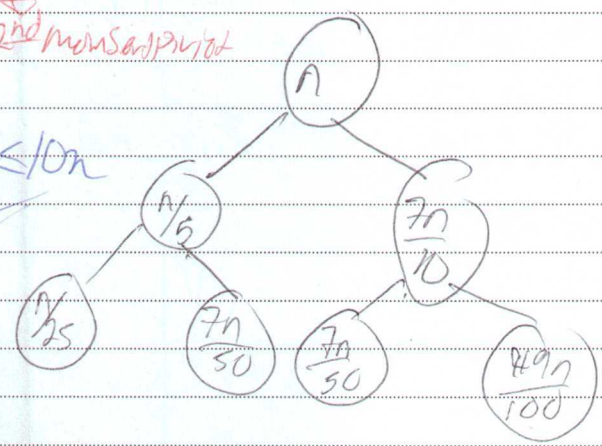
Recurrence:

$$T(n) \le T(n/5) + T(7n/10) + cn \in O(n)$$

(cn → Partitioning; T(n/5) → 1st MomSelectPivot; T(7n/10) → 2nd MomSelectPivot)

$$\Rightarrow T(n) \le n \sum_{i=0} \left(\frac{9}{10}\right)^i \le 10n$$

$$\in O(n)$$



Integer Multiplication

DC Q8
$O(n^2)$

```
 12      1100
x13     x1101
---     ----
 36     1100
 12x    0000
```

Division

Consider $x = x_1 \cdot 2^{n/2} + x_0$ & $y = y_1 \cdot 2^{n/2} + y_0$

$$xy = (x_1 \cdot 2^{n/2} + x_0)(y_1 \cdot 2^{n/2} + y_0) = \underbrace{x_1 y_1}_{1} \cdot 2^n + \underbrace{(x_1 y_0 + x_0 y_1)}_{2,3} \cdot 2^{n/2} + \underbrace{x_0 y_0}_{4}$$

4 bits 4 bits

x: 1010 1111 $= 1010 \cdot 2^4 + 1111$

x_1 x_0

$1234 = 12 \cdot 16^2 + 34$

DC Q9

How many recursive calls? 4

How big is the recursive call? $n/2$ - Splitting into hi / low bit

What is the cost per call? $O(n)$ linear

What is the recurrance?

$$T(n) \leq 4T(n/2) + cn$$

$$T(n) \leq 4T(n/2) + cn = O(n^{\log(4)}) \Rightarrow O(n^2)$$

Recursion

$P_1 = \text{int Mult}(x_1 + x_0, y_1 + y_0)$

$x_1 y_1 := \text{int Mult}(x_1, y_1)$

$x_0 y_0 = \text{int Mult}(x_0, y_0)$

Combine Return $x_1 y_1 \cdot 2^n + (P - x_1 y_1 - x_0 y_0) \cdot 2^{n/2} + x_0 y_0$

Because

$$(x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0 \equiv$$

$$x_1 y_0 + x_0 y_1$$

Recurrence $T(n) \leq 3T(n/2) + O(n) =$

$O(n^{\log(3)}) = O(n^{1.59})$

Finding the Closest Pair of Points

Problem

Given a set n points, $P = \{p_1, p_2 \dots p_n\}$ in plane. Find the closest pair.

That is Solve $\arg\min_{(i,j)}$

1-D: Closest Pair

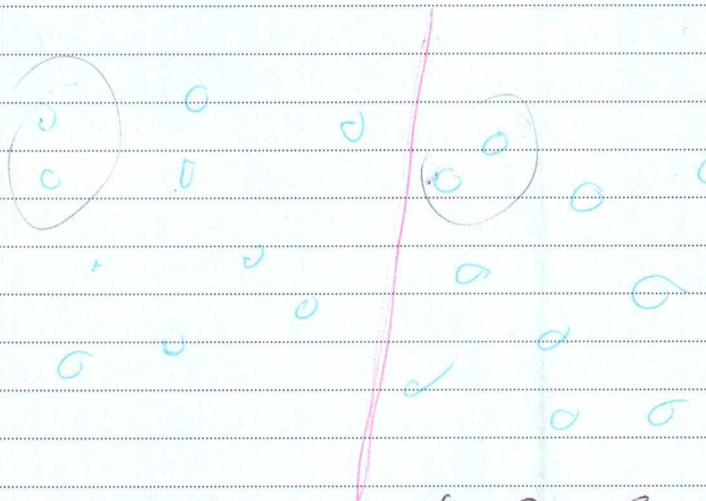
~~the closest~~ the points are on the line

- sort the points ($O(n \log n)$)
- walk through the sorted points and find minimum pair $O(n)$

Greedy ?

2-D. Divide / Conquer

- Divide: Split point set into (half-?)



P_x : point sorted by x-coordinate

P_y : Points sorted by y-coord

(Resp. R) is left (resp. right) . half P_x

- From P_x ^ P_y we create Q_x, Q_y, R_x, R_y w/o resorting.
- Runs in time $O(n)$
- Let (q_0^*, q_1^*) ^ (r_0^*, r_1^*) be closest pair in Q/R

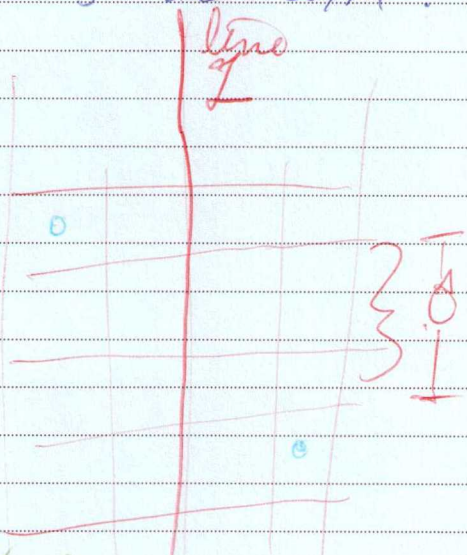
Let $\delta = \min\{d(q_0^*, q_1^*), d(r_0^*, r_1^*)\}$

If there exists a $q \in Q$ and an $r \in R$ for which $d(q,r) < \delta$, then each of q and r are within δ of L

L4 Q7

Lemma 3

Let S be the set of points within δ of L
- If there exists $s, s' \in S$ and $d(s,s') < \delta$ then s and s' are w/in 15 positions of each other in S_y



- partition δ-space R into $\delta/2$-squares
- At most 1 point per square else contradicts definition of δ.
- By way of contradiction say $d(s,s') < \delta$ and s, s' separated by 16 positions
- By counting argument s, s' are separated by 3 rows

Sorting $O(n \log n)$
2 recursive calls
$n/2$ size recursive call
$|S| = O(n)$ work per call

$T(n) \leq 2T(n/2) + cn = O(n \log n)$

Check All Subarrays

```
for i:1 to len(A)
    for j:i to len(A)
        if sum(A_i ... j) > sum
            M := A[i ... j]
```

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\Theta(n^2)$$

MaxSubArray
In Array A of n ints
Out max subarray in A
If $|A| = 1$ → return

A_1 = MaxSubArray(first-Half A)
A_2 = MaxSubArray(Back-half A)
M_1 = MidMaxSubArray(A)
Return Array with max sub of $\{A_1, A_2, m\}$

$O(n \log n)$

MidMaxSubArray
In Array of n ints
Out max subarray that crosses midpoint A.
m: mid point of A
L: max subarray in $A[i, m-1]$ for $i = m-1 \to i$ $O(n)$
R: max subarray in $A[m, j]$ for $j = m \to n$
return L U R combining L & R

- Correctness by induction

Complexity: same relation MergeSort

TÜL

(1950's)

"Dynamic Programming" - Richard Bellman

- Air Force
- UW Grad alum

Weighted Interval Scheduling

- requests $R = \{0, \ldots, n\}$
- request $r_i = (s_i, f_i, v_i)$ where s_i start, f_i finish time, v_i is value
- Goal: compatible schedule S has max value

$S: \forall r_i, r_j \in S: f_i \leq s_j \vee f_j \leq s_i$

- Assume R ordered by finish time
- Find the optimal value in subset R of first j items:
  - Find largest $i < j$ s.t. $f_i \leq s_j$
  - $OPT(j) = \max(OPT(j-1), OPT(i) + v_j)$

Proof of Optimality

- By strong induction

Base case: $j = 0$ (or $j = 1$) // only one possible sch.

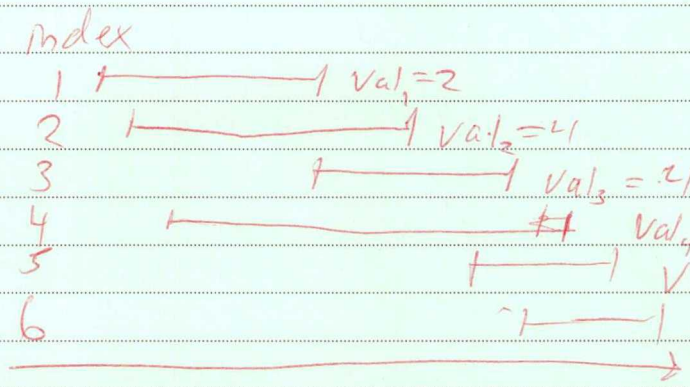
Inductive Step:

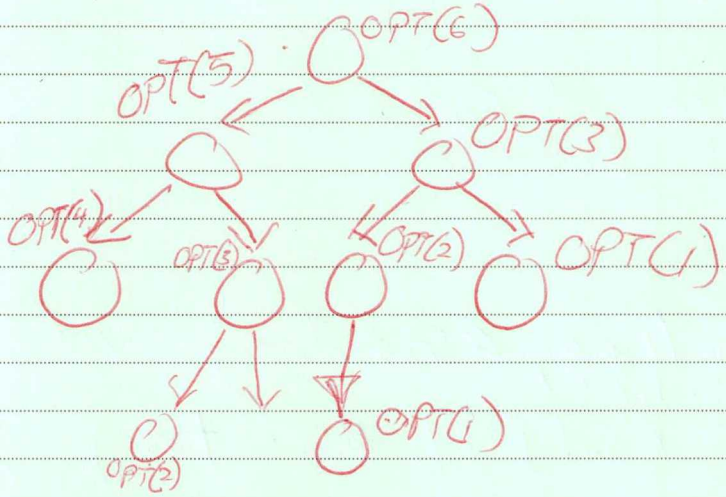
- By ind. hypothesis we have $j-1$ and Opt for i
- FF assures dichotomy, that the last interval is either in the solution or not
- take the max of whether or not a given interval is included

Consider the Recursion

$$OPT(j) = \max(OPT(j-1), OPT(i) + v_j)$$

$O(2^n)$





Memoization (1989) Donald Michie
"memorandum" → to be remembered

Weighted Int DP
Sort B by finish time
$M[0] := 0$
for $j := 1$ to n do
  Find index i
  $M[j] := \max(M[j-1], M[i] + v_j)$
end

↓
The optimal value for the schedule Q_j

TOL

Definition
- A sorted by finish time
- for a given job at index $j > i$
  - i is the largest index s.t. $f_i \leq s_j$

Description of matrix
- 1D array, where index j is the max value of a compatible schedule for the first j items in sorted A

Bellman Equation
- $M[j] = \max(M[j-1], M[i] + v_j)$

Location of solution, Order to populate
- The solution is found

DP Solution
- A sorted by finish time ascending order
- for a given job at index j to i
- Bellman EQ: $M[j] = (M[j-1], M[i] + v_i)$

Runtime
- Preprocessing $O(n \log n)$

Populate
cells $O(n)$
cost per cell $O(n)$

Overall $O(n^2)$ linear search $O(n \log n)$ binary search

Q1 (2^n) 10/18/22 Q3 (2^n) Q4 (n^2)

Q5 (2) Q2 → ∞, A[1...n] | distinct cells

algorithm LIS:
Input: integer k, an array of Int A[1...n]
Output: Return length of LIS where every value > k. ex: complete the algorithm

~~Base case (|A|) → return 0~~

if n=0 then return 0 — Base
else if A[1] ≤ k then
  return LIS(k, A[2...n]) — A[1] not ∈ LIS
else — A[1] > k maybe
  skip: LIS(k, A[2...n]) → ¬A[1]
  ~~skip~~ take: LIS(A[1], A[2...n]) + 1 → ∈ A[1]
  return
end

Natural Dichotomy — assume A is the starting player

Heads / Tails

$$\max\left\{\left(c_i + B^*_{c_{(i+1)\dots j}}\right), \left(c_j + B^*_{c_{i\dots(j-1)}}\right)\right\}$$

$$B^*_{c_{i\dots j}} : \min\left\{\left(A^*_{c_{(i+1)\dots j}}\right), \left(A^*_{c_{i\dots(j-1)}}\right)\right\}$$ (heads / tails)

In this game Bob minimizes the payout to Alice

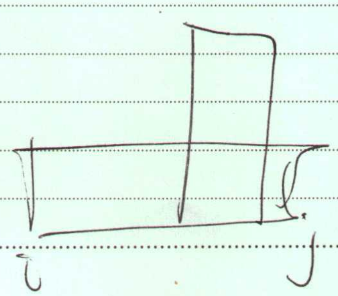
2D Array:

M_{ij} is the max value possible for Alice when choosing from $c_{i\dots j}$ for Bob*

Bellman Eq:

$$M_{ij} = \max\left\{ c_i + \min\{M_{i+2,j}, M_{i+1,j-1}\},\ c_j + \min\{M_{i+1,j-1}, M_{i,j-2}\}\right\}$$

game 4 $O(n^2)$



proof of correctness / strong induction

Solution is in cell M[1, n]

$O(n^2)$

$$Start_i = \begin{cases} Start_{i-1} & s_{i-1} + a_i > a_i \end{cases}$$

if $n \notin S \rightarrow r_n = r_{n-1}$

1D approach (Flawed)

if $n \in S$ then $r[n] = V[n-1]$

if $n \notin S$ then $r[n] = ?$

Keeping n does automatically exclude other items

To solve: best solution with $n-1$ previous items restricted by w

- Best solution with n, previous items restricted by $W - w_n$

2D approach

i: items indices 0 to n

w: max weight 0 to w

$x_{iw} := 0$ if $w_i > w$ and 1 otherwise

$V[i,w] = \max(V[i-1,w], x_{iw} \cdot (V[i-1, w-w_i] + w_i))$

$V[0,w] := 0 \ \forall w$ & $V[i,0] = 0 \ \forall i$

Solution value $[n, w]$

Edit Distance

Minimum Number of Letter
- insertions : adding a letter
- deletions : removing a letter
- substitutions : replacing a letter

to change String $A[1...n]$ to String $B[1...m]$
E.g. Tuesday → Thursday → Thursday

Let $A[1...n]$ & $B[1...m]$ be to input strings
- What if we edit distance for $A[1...i]$ / $B[1...j]$
  - insertion: $Edit(i,j) = Edit(i,j-1) + 1$
  - Deletion: $= Edit(i,j) = Edit(i-1,j) + 1$
  - Substitution: Edit $(A[i] \neq B[j])$ Penalty

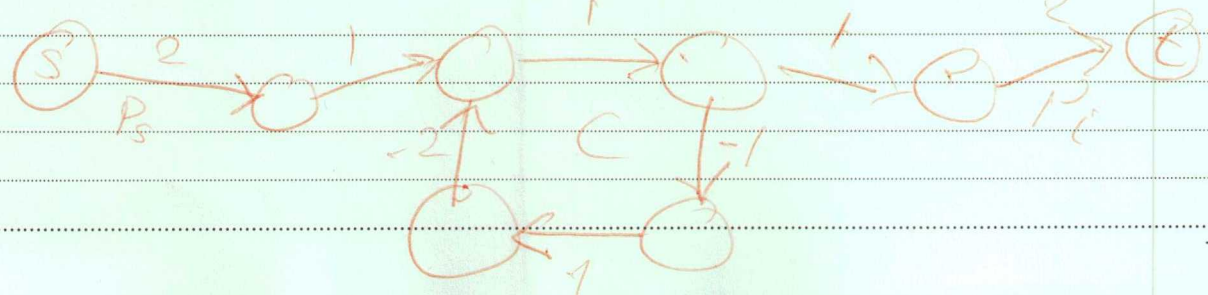
Base Case
$i = 0$, $Edit(i,j) = j$
$j = 0$, $Edit(i,j) = i$

2 Dimensions

$$E[i,j] = \begin{cases} i, & j = 0 \\ j, & i = 0 \\ \min\{E[i,j-1]+1,\ E[i,j-1]+1,\ E[i-1,j-1] + (A[i] \neq B[j])\} \end{cases}$$

We have a DAG $G = (V, E)$ where $|V| = n$ and $|E| = m$ and a node s that has a path to every node in V. For each edge $e = (i,j)$, c_{ij} is the weight of the edge, and there are no cycles with negative weight.

- What is the shortest path from s to each other node?



Observation 1

If G has no negative cycles then there exists a shortest path from s to t that is simple (have no cycles) and has at most $n-1$ edges

D.P. : 2D matrix M of edges in path x vertices

$M_{i,v}$ is the shortest path from v to t using $\leq i$ edges

Solution is in $M_{n-1,s}$

Dichotomy:

use $\leq i-1$ edges

use $\leq i$ edges

$$M_{i,v} = \min\left\{ M_{i-1,v}, \min_{w \in v}\left\{ M_{i-1,w} + c_{vw} \right\} \right\}$$

where $c_{vw} = \infty$

$O(n^2)$ complexity, $O(n)$ cost per cell

overall $O(n^3)$

Worst Case: n Nodes, m edges

- For each node v, we only need to consider out going edges to w (at most n)
- For every node v, we need to do this calculation for $0 \leq i \leq n-1$ lengths

Overall $O\left(n \sum_{v \in V} n_v\right) = O(mn)$

TOL

Observation 2

If there is a negative cycle along the path from s to t then the shortest path is $-\infty$

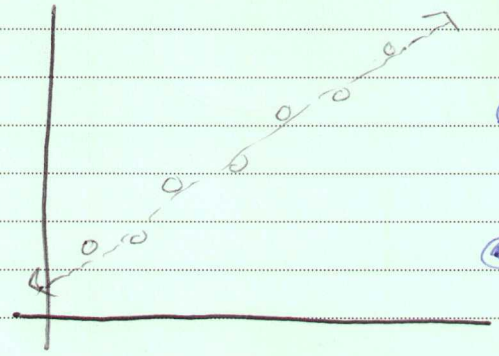
Observation 3

$M[i,j] = M_{n-1,v}$ $\forall i, i > n-1$ and $\forall v \in$ Nodes if there are no negative cycles on the path to t.

Augmented Graph for Negative Cycle finding

- add a node t w/ an incoming edge from all the nodes with cost 0
- run Bellman-Ford from any node s to t until number of edges n
- if for some v, $M[n][v] = M[n-1][v]$ then there is a negative cycle

---



we want to find interpolation

setup:

- $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ on the plane
- $x_1 < x_2 < \ldots < x_n$

(1) partition the points (by x) into contiguous subsets

(2) minimize the sum of $Error(I, P_i) + C$ for all subsets, where C is a fixed cost per subset

- find I: $y = ax + b$ that minimizes $Error(I, P_i) = \sum_{i=1}^{n} (y_i - ax_i - b)^2$

$S_j = \min_{i \le j} (e_{i,j} + C + S_{i-1})$

preprocess: $O(n^2)$, # of cells $O(n)$

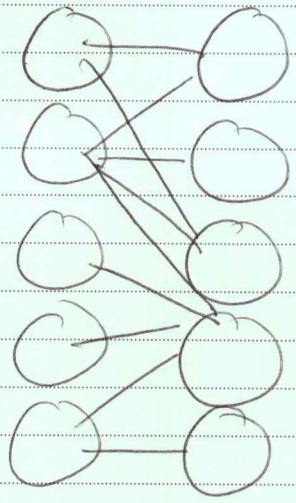
work done for cells $O(j)$ overall $O(n^2)$

e_{ij} is the min error for partition from i to j

C is added each time

S_i is optimum up to point i

Bipartite Matching Problem



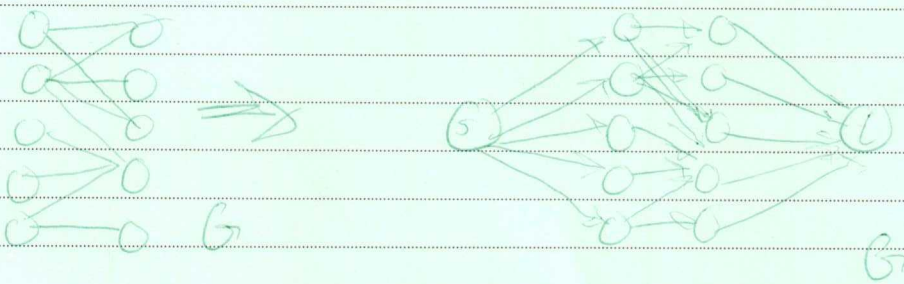
Definition

- Bipartite Graph. $G = (V = X \cup Y, E)$
- all edges go between X & Y
- matching $M \subseteq E$ s.t. Node appears in only one edge
- Goal: Find largest matching (cardinality)

Reduction to Max-Flow Problem

Goal: create a flow Network based on the original problem

- The solution to the flow network must correspond to the original problem
- The reduction should be efficient



Edge-Disjoint Paths

Problem

given a graph $G = (V, E)$ and two distinguished nodes s and t find the number of edge-disjoint paths from s to t.

Flow Network

- Directed Graph
  - S.P the source s is the source, t is the sink
  - add capacity of 1 to every edge
- Undirected Graph
  - For each undirected edge (u, v), convert to 2 directed edges (u, v) and (v, u)
  - apply directed graph transformation

Observation

If there are k edge-disjoint paths in G from s to t then the max-flow in G' is k.

Runtime — Basic FF Method $(m C) = O(mn)$

Path Decomposition

1.) Let f be a max-flow for this problem.
We can find k edge-disjoint paths.
- DFS from s on f along edges e where $f(e) = 1$.
  1) find a simple path P from s to t
  → set flow to 0 along P; continue DFS from s.
  2) find a path P with a cycle C before reaching t; set flow to 0 along C. — continue DFS start of cycle

Image segmentation

Problem

Let P be the set of pixels in an image. We would like to separate P into set A and B, where A are the foreground pixels and B are the 'background pixels'.

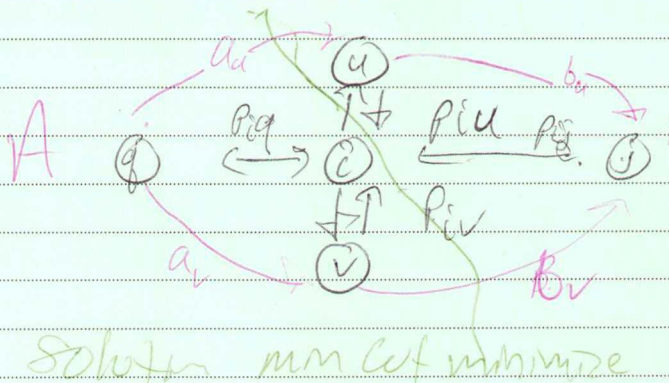
For pixel i:
- $a_i \geq 0$ – likelihood of i being in A
- $b_i \geq 0$ – likelihood of i being in B
- for each adjacent pixel j: $p_{ij} = p_{ji}$ is a separation penalty paid when i and j are not both $\in A$ or $\in B$

Goal:
- Maximize $q(A,B) = \sum_{i=0}^{A} a_i + \sum_{j=0}^{B} b_j - \sum_{i,j \in P' | A \cap \{i,j\} | = 1} p_{ij}$

Fact: if $Q = \sum_{i=0}^{P} a_i + b_i$

then $q(A,B) = Q - \sum_{i=0}^{B} a_i - \sum_{j=0}^{A} b_j - \sum_{i,j \in P' | A \cap \{i,j\} | = 1} p_{ij}$

↳ minimize $\sum_{i=0}^{B} a_i + \sum_{j=0}^{A} b_j + \sum_{i,j \in P' | A \cap \{i,j\} | = 1} p_{ij}$



Reduction:
1.) each pixel becomes node
2.) add edges between neighbors i,j – capacity p_{ij}
3.) add source s and connect to all nodes i with capacity a_i
4.) add sink t and connect all nodes i with capacity b_i to t

Solution: min cut minimize $\sum_{i \in B} a_i + \sum_{i \in A} b_i + \sum_{i,j \in P \text{ s.t. } |A \cap \{i,j\}| = 1} p_{ij}$

- $i \in A$ – foreground contributes b_i to cut
- $i \in B$ – background node contributes a_i to cut
- $i \in A, j \in B$ – i,j edge cut → p_{ij} to cut

Flow Network Extension

11 | 10 | 22

Flow demand Network

- Each node has a demand d_v
  - if $d_v < 0$: a source that demand $f^{in}(v) - f^{out}(v) = d_v$
  - if $d_v = 0$: internal node ($f^{in}(v) - f^{out}(v) = 0$)
  - if $d_v > 0$: a sink that demands $f^{in}(v) - f^{out}(v) = d_v$
- S is set of sources
- T is set of sinks

Flow Conservation

- Capacity: for each $e \in E$, $0 \le f(e) \le c_e$
- Conservation: for each $v \in V$, $f^{in}(v) - f^{out}(v) = d_v$

Goal

Feasibility: Does there exist a flow satisfies all conditions?

Lemma 10: (not iff)

if feasible flow exists then

$$\sum_{v \in V} d_v = 0$$

Proof:

Suppose f is feasible flow, then by definition, for all v, $d_v = f^{in}(v) - f^{out}(v)$

For every edge $e = (u, v)$, $f_e^{out}(u) = f_e^{in}(v)$. Hence $f_e^{in}(v) - f_e^{out}(u) = 0$

Hence $\sum_{v \in V} d_v = 0$ which is what we wanted for flow.

Corollary 11

if there is a feasible flow then

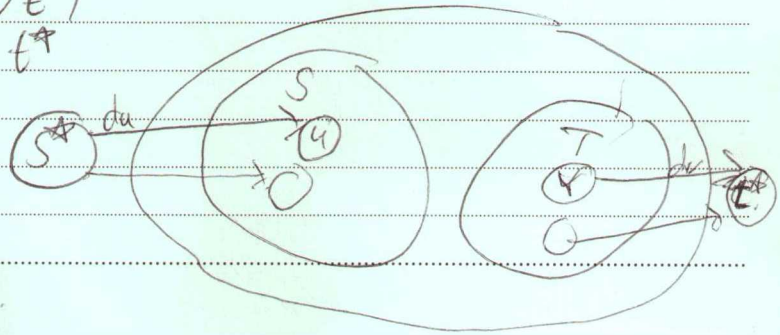
$$0 = \sum_{v : d_v > 0 \in V} d_v = \sum_{v : d_v < 0 \in V} -d_v$$

Reduction to Max flow (from G [demands] to G' [no demands])

- Super source s^*: Edges from s^* to all $v \in S$ with $d_v < 0$ with capacity $-d_v$
- Super sink t^*: Edges from all $v \in T$ with $d_v > 0$ with capacity d_v to t^*
- maximum flow of $D =$

$$= \sum_{d_v > 0 \in V} d_v = \sum_{v : d_v < 0 \in V} -d_v$$

in G' shows feasibility



TÜL

Another Flow Network Extension

11/10/22

Adding lower bounds

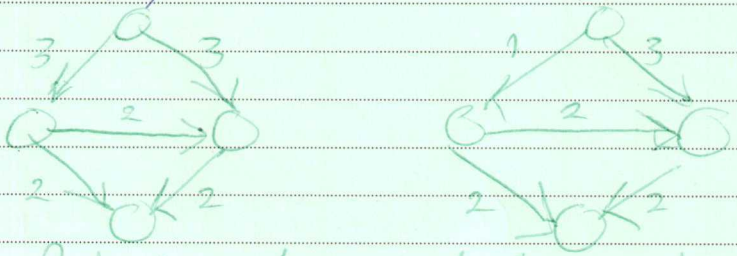
- For each edge e, define a lower bound l_e, where $0 \le l_e \le c_e$

Flow Conditions

i Capacity: for each $e \in E$, $l_e \le f(e) \le c_e$

ii Conservation: for each $v \in V$, $f^{in}(v) - f^{out}(v) = d_v$

Goal → Feasibility: Does there exist a flow that satisfy conditions?



$f_0((u,v)) = 2$

$l_v = -2$

$l_u = 2$

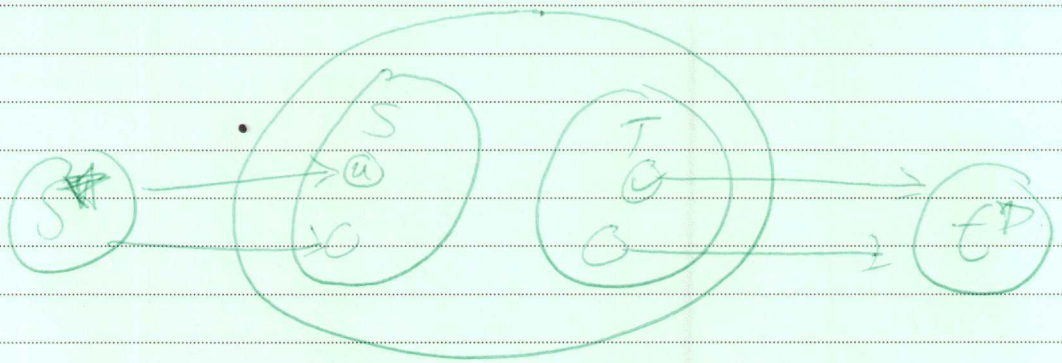
Step 1: Reduction from G (demand + LB) to G' (demand)

- Consider an f_0 that sets all edge flows to l_e:
  $L_v = f_0^{in}(v) - f_0^{out}(v)$
  - if $L_v = d_v$: condition satisfied
  - if $L_v \neq d_v$: imbalance
- For G'
  - Each edge e, $c'_e = c_e - l_e$ & $l_e = 0$
  - Each node v, $d'_v = d_v - f_v$

Step 2: Reduction from G' (demand) to G'' (no demand)

- Super source s^*: Edges from s^* to all $v \in S$ with $d_v < 0$ with capacity $-d_v$.
- Super sink t^*: Edges from all $v \in T$ with $d_v > 0$ with capacity d_v to t^*
- Maximum flow of $D = \sum_{v: d_v > 0 \in V} d_v = \sum_{v: d_v < 0 \in V} -d_v$ in G'

This shows feasibility!



Survey design

Problem

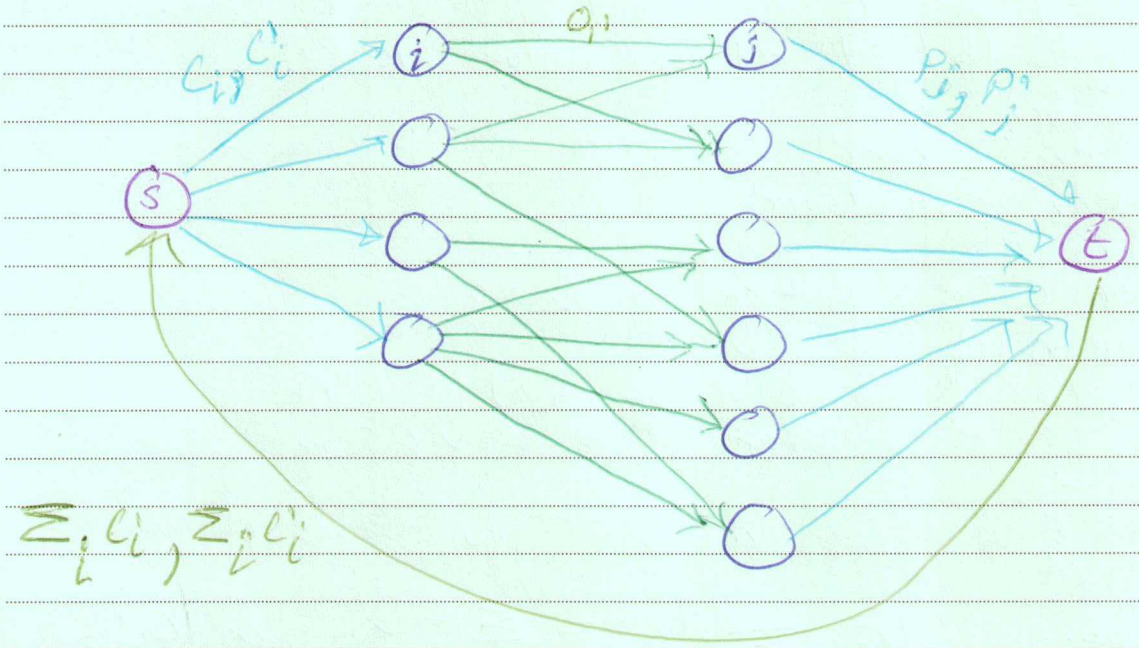
- Study of consumer preferences
- A company, with K products, has a database of n customer purchase history

Goal: define a product specific survey

Survey Rules

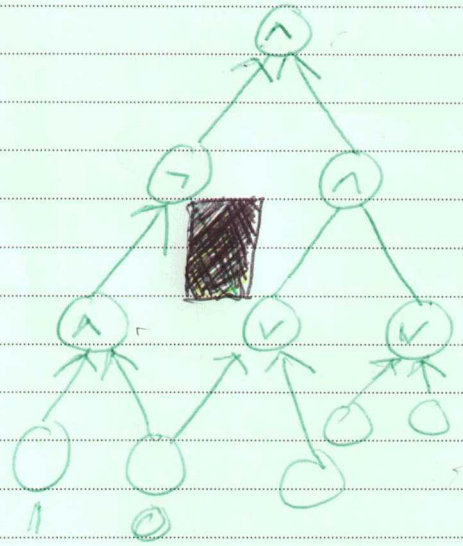
- Each customer recieves a survey based on their purchases
- Customer i will be asked about at least c_i and at most c'_i products
- To be useful, each product must appear in at least p_i and at most p'_i surveys

Bipartite graph (Customer[i], Products[j])



Circuit Satisfiability (CSAT)

Problem definition
- 3 types of gates: $\wedge$, $\vee$, $\neg$
- Circuit K:
  - A DAG (nodes 0, 1) or R in comm. edges
  - Source: nodes with no incoming edges
  - every other node is labelled with a gate
  - output result of top operand node is 1

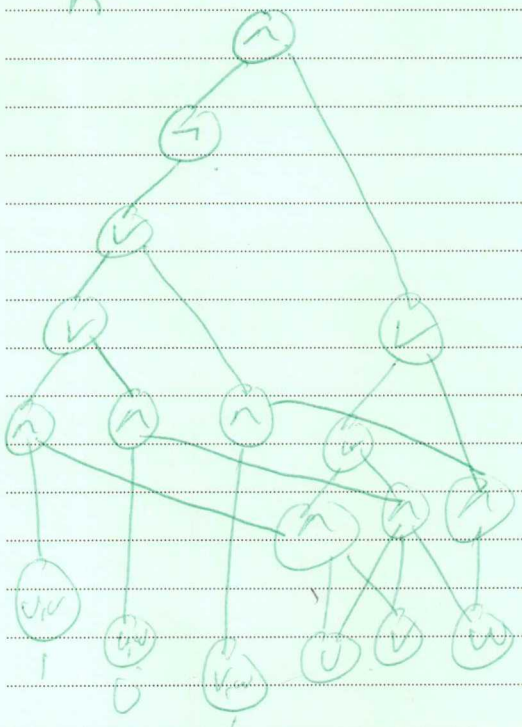


Theorem 6
– Cook (1971) – Levin (1973)

Show CSAT $\in$ NP
- input size $\Omega(|V|)$
- single gate is evaluated in constant time
- Evaluate a certificate of the inputs can be verified in $O(|V|)$

★ Reduce every problem $\in$ NP to CSAT
- consider $X \leq_p$ CSAT
- by definition for X:

K=2



– Source: $|S| + |t| = n + \text{poly}(n)$ bits
– the first n bits are hard-coded to the X-instance input
– the poly(n) bits are free and used to find a t

Cook Reduction
- prove $X \in$ NP
- choose problem $Y \in$ NP-complete
- prove $Y \leq_p X$

Karp Reduction
1. provide efficient reduction
2. prove if S_Y is true S_X is true
3. prove if S_X is true S_Y is true

For an arbitrary circuit K

- each node v is assigned a variable x_v
- for each gate
  - ¬: Let u be the input
    we need $x_v = \overline{x_u}$
    so: $(x_v \vee x_u) \wedge (\overline{x_v} \vee \overline{x_u})$
  - ∨: Let u, w inputs. We need $x_v = x_u \vee x_w$
    $(x_v \vee \overline{x_u}) \wedge (x_v \vee \overline{x_w}) \wedge (\overline{x_v} \vee x_u \vee x_w)$
  - ∧: Let u, w be inputs. We need $x_v \wedge x_w$
    $(\overline{x_v} \vee x_u) \wedge (\overline{x_v} \vee x_w) \wedge (x_v \vee \overline{x_u} \vee \overline{x_w})$
- for each constant source s_i
  x_{s_i} if 1 and $\overline{x_{s_i}}$ if 0
- for the output o_i: 1 clause (x_{o_i})

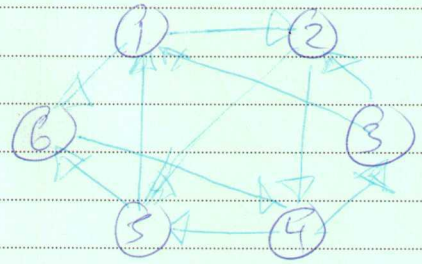
Finally, Convert Clauses to Length 3

Taxonomy - Traveling Salesperson (TSP)

- a salesperson must visit n cities
  $v_1, v_2, \ldots, v_n$
- Starting at some v_1, visit all cities and return to v_1
- Distance function $d(i, j)$ for all pairs of cities

Hamiltonian Cycle

- Graph analogue of TSP
- Hamiltonian cycle: a tour of length of G that visits each node once



{1, 6, 4, 3, 2, 5}

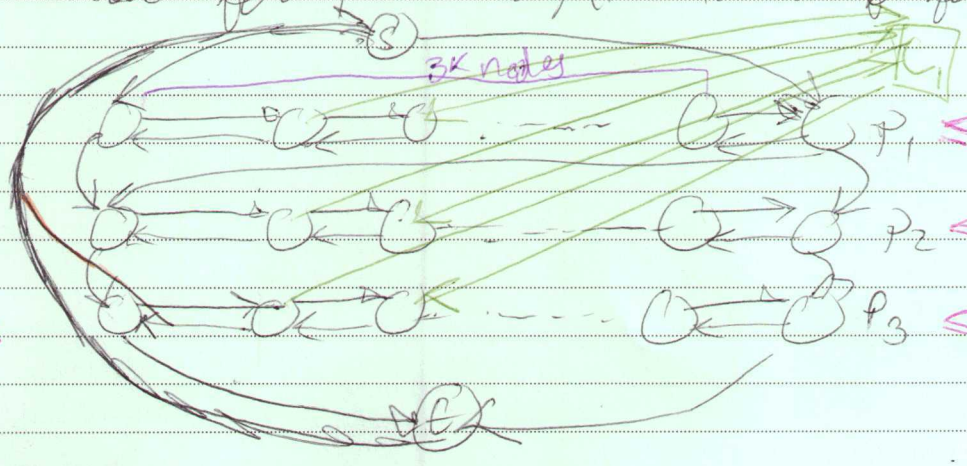
Proof

- In NP: a certificate that would be a sequence of vertices which can be verified in polynomial time
- choose an NP-complete problem 3SAT
  (Shown NP Complete)

Known → what we want to show

3SAT $\leq_p$ Hamiltonian

- for (contains 3k + 2 nodes) for each x_i left traversal for 1 and right traversal for 0



Theorem 9 (TSP) is NP-Complete

(1) In NP: certificate that is a tour of the G? Yes
(2) Use Hamiltonian Cycle
(3) Hamiltonian Cycle $\leq_p$ TSP
(4) Given Graph $G = (V, E)$
- for each v, make a city
- for each edge $(u,v) \in E$ define $d(u,v) = 1$
- for each pair $(u,v) \notin E$ define $d(u,v) = 2$
- set the bound to be n
- $\Leftarrow$

Theorem 10 - Hamiltonian Path is NP-Complete

(1) In NP: certificate that
(2)
(3) Hamiltonian Cycle $\leq_p$ Hamiltonian Path
For $G = (V, E)$ create G'
- Choose an arbitrary $v \in V$; $V' = V \cup \{v_1, v_2\}$
- $E' = E \cup \{(v_1, v), (v_2, v)\}$
- for each

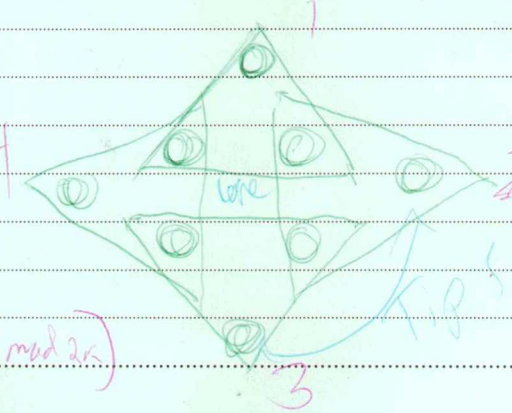
3D-Matching

- Given 3 disjoint sets X, Y, Z (each of size n)
- A set of $m \geq n$ triples $T \subseteq X \times Y \times Z$
- Does there exist a set of n triples from T so that each item is in exactly

Theorem 11 - 3D-Matching is NP-complete

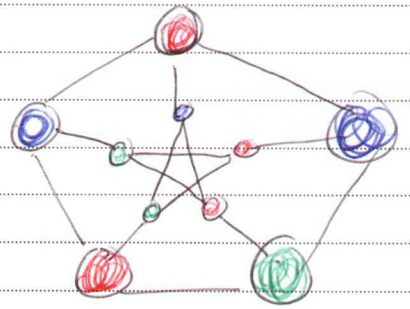
Variable x_i gadget.

Let:
- $A_i = \{a_1^i, \ldots a_{2k}^i\}$
- $B_i = \{b_1^i, \ldots b_{2k}^i\}$
- $t_j^i = (a_j^i, a_{j+1}^i, b_j^i)$ for $j = 1, 2, \ldots 2k$ (add mod $2k$)



always selecting a triple

Graph Colouring



Given a graph G and bound k
Does G have k-colouring

k-Colour
- a colouring of the nodes of a graph such that no adjacent nodes have the same colour
- Labelling (partition) function
  $f: V \to \{1, \dots, k\}$ s.t.
  $\forall (u,v) \in E \; f(u) \neq f(v)$

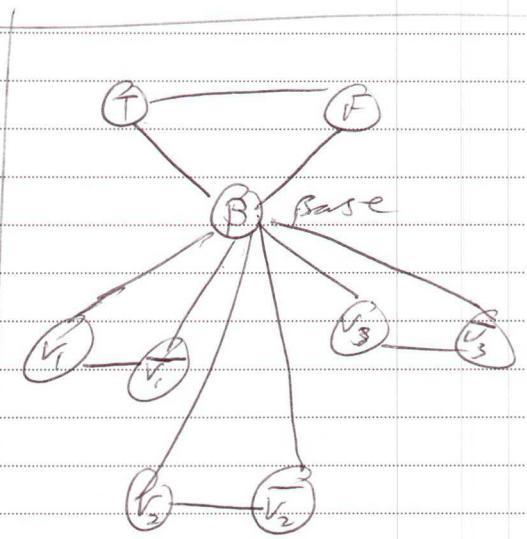
2-Colour
$\to$ Bipartite

First prove the solution is in NP
that is Show $f(u) \neq f(v) \; \forall (u,v)$ $O(|V_i| + |V_j|)$
So 3-colouring $\in$ NP

Use the fact that 3-Sat is NP-complete to show that 3-colouring is at least as hard as w/ reduction to prove 3-colouring is also NP-complete
That is prove

$$3SAT \leq_p 3\text{-Colouring}$$

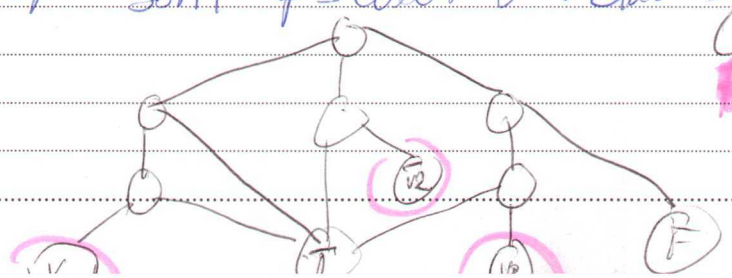
Since 3SAT $\in$ NP-complete
$\to$ 3SAT $\in$ NP-Hard
3Col $\leq_p$ 3SAT
$\Rightarrow \forall P \in NP \leq_p$ 3SAT
Show 3SAT $\leq_p$ 3Col
$\forall P \in NP \leq_p$ 3Col



Each literal V_i and $\overline{V_i}$
Nodes True(true), false(F), and Base
Edges: $(V_i, \overline{V_i})$, (V_i, B), $(\overline{V_i}, B)$.
Edges: (T,F) (F,B), (T,B)
// T = F = B all different colour

Boolean variable Gadget

Theorem 12 3-Colouring is NP complete
Proof 3SAT $\leq_p$ 3 colouring + clauses



Set up
Clauses: e.g. $C_1 = (x_1 \vee \overline{x_2} \vee x_3)$
V_1, V_2, V_3
one must be true
to satisfy the prob

TÜL

Numerical Problem - Subset Sum Problem

Given set of n natural numbers $\{w_1, \ldots, w_n\}$ and target W is there a subset of the numbers that add up to W

recall the DP example

$O(nW)$ algo

W is unbounded 2^n

Theorem - Subset Sum is NP-complete

(1) · NP - Certificate ↪ the problem's solution is verified in poly time

(2) Use 3D matching to show Subset Sum is at least as hard as 3D matching by reducing Subset Sum to 3D-matching problem. Since 3D matching is already proven NP-~~Hard~~ complete

Recall 3D matching is NP-complete

≠ 3D-Matching ≤p Subset Sum

- 3D-matching subset can be viewed as length $3n$ bit vector with a 1 indicating that item is in the set
- for each triple (i, j, k) from $X \times Y \times Z$ construct a w_t
  - a digit with 1 @ i, $n+j$ and $2n+k$
  - for base d $w_t = d^{i-1} + d^{n+j-1} + d^{2n+k-1}$
  - set base $d = n+1$ to avoid carry over

// odd

// padding

suppose base $d = 2$

$2^{i-1} + 2^{n+j-1} + 2^{2n+k-1}$

$d = 10$: $10^{i-1} + 10^{n+j-1} + 10^{2n+k-1}$

To finish the reduction

- each digit happens exactly once

iff $\sum_0^{3n-1} (n+1)^i$ ⟺ $3n$ bit vectors

Set $W = \sum_0^{3n-1} (2n+1)^i$

~~Plat~~ Taxonomy — 12 / 1 / 22

Packing (maximizate)
- Independent set
- set packing

Covering (collection from universe) // minimize
- vertex cover
- set cover

Sequencing problem
- TSP
- Hamiltonian cycle
- Hamiltonian path

Partition
- 3D matching
- Graph coloring

Numerical problem
- subset sum
- knapsack

Constraint satisfaction problem
- 3SAT
- CSAT

Set ... certificate

1. Verify NP — $h(x) \in NP$
2. Comparative problem?
2a. 3SAT
3. Reduction ($3SAT \le_p h(x)$)
4. Prove by condition

Inpt ⟷ Sol

Sol ⟶ Inpt

3D ⟶ 4D matching

$3SAT \le_p h(x)$

$h(x)$ is at least as hard as 3SAT

@ coNP

Assymetry of NP

Efficient certifier assymetry

Given an instance s of problem X:
- for any t, $B(s,t) = yes \xrightarrow{implies}$ yes-instance
- for any t, $B(s,t) = no \longrightarrow$ no-instance

Complimentary problem
- $\forall$ problem X 有 $\overline{X}$
- $\forall$ input s, $s \in X \longleftrightarrow s \notin \overline{X}$
- since $X \in P \rightarrow \overline{X} \in P$

Thus a problem $X \in coNP \leftrightarrow \overline{X} \in NP$ since theorem 14

Complexity class NP

a problem $X \in coNP \longleftrightarrow \overline{X} \in NP$

Theorem 14

$NP \neq coNP \rightarrow P \neq NP$

Proof
- $X \in NP \rightarrow X \in P \rightarrow \overline{X} \in P \rightarrow \overline{X} \in NP \rightarrow X \in coNP$
- $X \in coNP \rightarrow \overline{X} \in NP \rightarrow \overline{X} \in P \rightarrow X \in P \rightarrow X \in NP$

is $P = NP \cap coNP$?

is $NP = coNP$?

cert has to be poly w.r.t. input

P - Space (Beyond Time)

Set of all problems that can be solved using polynomial space

Theorem 15: $P \subseteq PSPACE$

Theorem 16: $NP \subseteq PSPACE$

Proof

- for 3SAT a bit vector can encode an assignment
- We try all bit vectors (with one n-length vector in memory)
  - start w/ 0s until $2^n - 1$
    - add 1 @ each iteration
- Since 3SAT $\in$ PSPACE and is NP-complete
  - $\forall Y \in NP, Y \leq_p 3SAT$ and solve in PSPACE

Prototypical PSPACE Problem "Contingency planning"

Let $\Phi(x_1, \ldots, x_n)$ be conjunction of K disjunctions of n variables (like SAT)

Quantified SAT

- $\exists x_1 \forall x_2 \exists x_3 \ldots Q_n x_n \Phi(x_1, \ldots, x_n)$
  - Prenex normal form

∴ theorem 17 - QSAT is PSPACE-complete ∎

TÜL

for $0 < \varepsilon < 1$

$T(n) = T(\varepsilon n) + T((1-\varepsilon)n) + \Theta(n)$

$= \Theta(n \log n)$

probabilistic Argument

$T(n) \leq \Pr[\Theta(n) \text{ split}] \cdot \Theta(n \log n) + \Pr[\Theta(n) \text{ split}] \cdot \Theta(n^2)$

$= (1 - \Pr[\Theta(n) \text{ split}]) \cdot \Theta(n \log n) + \Pr[\Theta(n) \text{ split}] \cdot \Theta(n^2)$

$= \Theta(n \log n) + \Pr[\Theta(n) \text{ split}] = 0 + \left(\frac{\log n}{n}\right)$

Average Case Recurrence (Given uniform distro. as recurs.)

$T(n) \leq \frac{1}{n} \sum_{i=1}^{n} \cdot (T(i-1) + T(n-i)) + \Theta(n)$

Harmonic series

one of the proofs

$= 2/n \sum_{i=1}^{n} (T(i-1) + \Theta(n)) = \Theta(n \log n)$

Uniformly At Random (UAR)

- Under distribution π
- Means random decisions
- Average case analysis: input considered non-deterministic

Given for some dist π

Monte Carlo

- with probability p returns correct ans
  - on multiple tries increase chance of correct ans
  - Provide an approximation guarantee in expectation

Las Vegas

- always return correct solution (when solution is returned)
- has polynomial in expectation

Algorithm OPT

- Probabilistic run-time and correctness

Guarantee in Expectation $= E$.

- returns a solution that has an approximation ratio in expectation

$\forall I, E_{\text{ALG}}[\text{ALG}(I)] \leq r \cdot \text{OPT}(I) + \eta$

Probability Space

Sample Space Ω of all possible outcomes
- can be infinite
- i.e. $\Omega = \{1, 2, 3, 4\}$

$i \Rightarrow P(i) \geq 0$

- probability mass: each $i \in \Omega$ has non-neg prob
- Total mass

$$\sum_{i \in \Omega} P(i) = 1$$

Probability Event

- an event E is a set of outcomes of Ω

$$Pr[E] = \sum_{i \in E} P(i) \qquad Pr(E) = 1 - Pr(\bar{E})$$

$$Pr[E|F] = \frac{Pr[E \cap F]}{Pr[F]}$$

Independent events

Events E and F are independent if

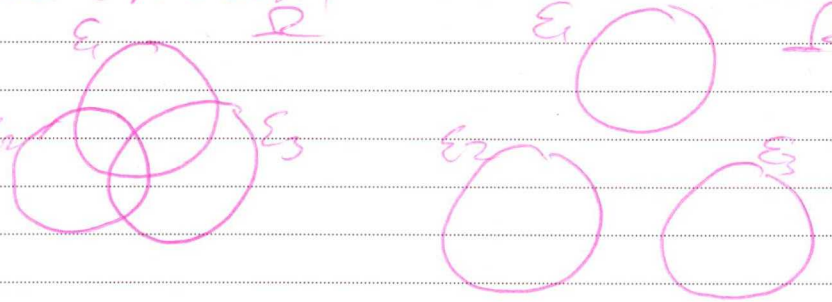
$Pr[E|F] = Pr[E]$ & $Pr[F|E] = Pr[F]$

$\Rightarrow Pr[E \cap F] = Pr[E] \cdot Pr[F]$

Suppose $E_1, E_2, \ldots E_n$ are independent

$$\Rightarrow Pr\left[\bigcap_{i=1}^{n} E_i\right] = \prod_{i=1}^{n} Pr[E_i]$$

Union Bound



$$\Rightarrow Pr\left[\bigcup_{i=1}^{n} E_i\right] \leq \sum_{i=1}^{n} Pr(E_i)$$

Expected value

$E(X) = \sum_{j=c}^{\infty} j \cdot Pr[X=j]$

4 sided dice

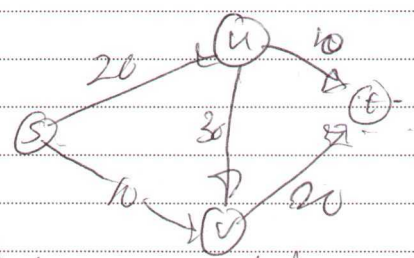
$\sum_{j=1}^{4} j \cdot Pr[X=j] = 1 \cdot \frac{1}{4} + 2 \cdot \frac{1}{4} + 3 \cdot \frac{1}{4} + 4 \cdot \frac{1}{4}$

$= 2.5$

input: array $A[1..n]$

output: sorted $A[1..n]$

Random min cut



global min cut = 0

Expected Runtime

VAR

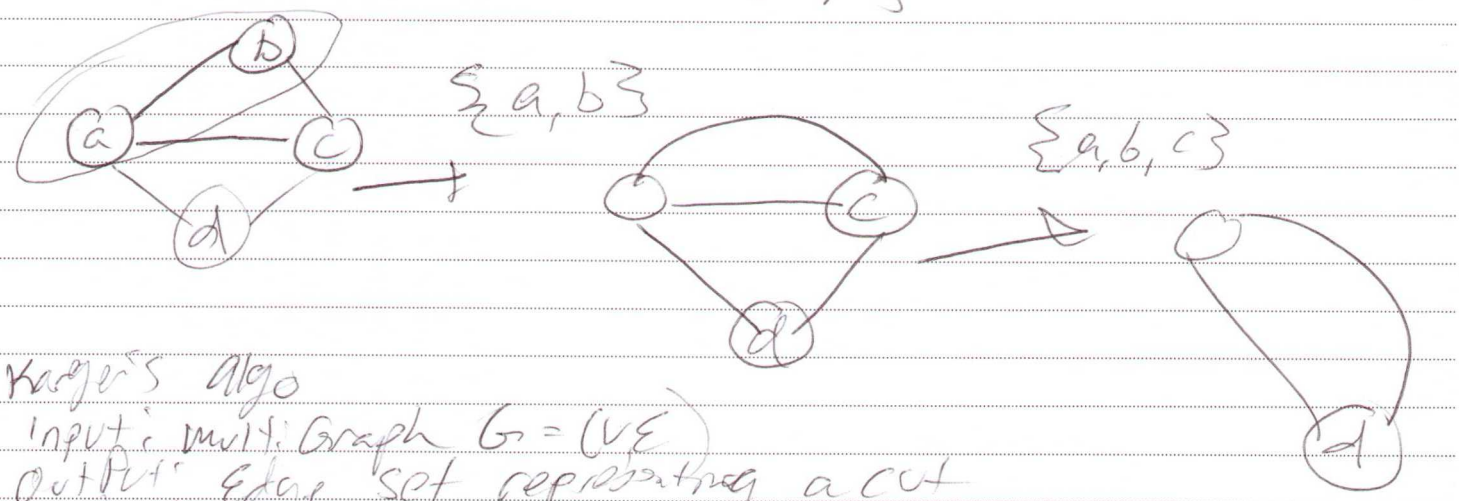
$T(n) \leq \frac{1}{n}$

- A Cut - Partition of V into sets (A, B) w/ $s \in A$ and $t \in B$
- Cut capacity: $c(A,B) = \sum_{e \text{ out}} A \subseteq e$
- minimum-cut of G: The cut (A^*, B^*) that maximizes $c(A^*, B^*)$ for G

Global min Cut

~~∀~~ pair (s,t)

- undirected multi-graph $G = (V, E)$
  - E is a multi-set (u, v) might be in E many times
- (u, v) edge contraction
  - create super-node $\{u, v\}$



Karger's algo

Input: Multi Graph $G = (V, E)$

Output: Edge set representing a cut

if: G has exactly 2 nodes $u \& v$

then → Return the set of edges between $u \& v$

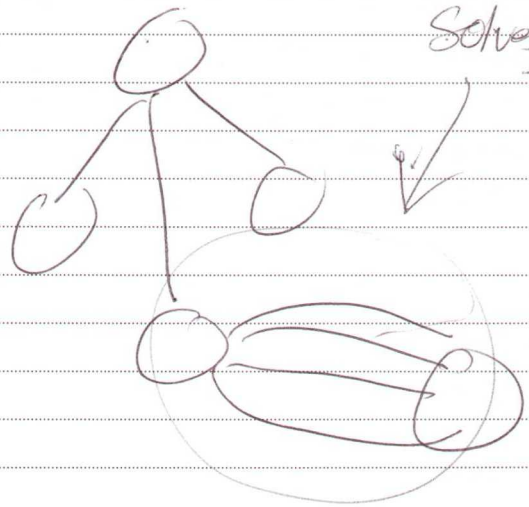
else

choose an edge (u, v) uniformly at random

$G' :=$ graph contracting (u, v)

Return (Contraction algo G')

end



Analysis of Kruger's Algorithm

Theorem 1: (The contraction algorithm returns a global min-cut of G w/ probability of at least $1/\binom{n}{2}$)

- Suppose the global min-cut (A, B) has a size of k, and let F be the edge set
- If every node has a degree $\geq k \rightarrow |E| \geq \frac{1}{2}kn$
  or $\geq k \rightarrow |E| \geq kn$
- $\Pr[\text{edge in } F \text{ is contracted in } (\text{Node}_1, \text{Node}_2) \text{ in } 1 \text{ step}] = \frac{k}{\frac{1}{2}kn} = \frac{2}{n}$

$$\leq \frac{k}{\frac{1}{2}k(n-(i-1))} = \frac{2}{n-i+1}$$

i conditioned on no edge from having been previously contracted

- There are $n-2$ steps in contraction algo
- Let i be the event that an edge $\in F$ is not contracted at step i.

$$\Pr[\text{success}] = \Pr[\bar{\varepsilon}_2 \mid \bar{\varepsilon}_1] \cdots \Pr[\bar{\varepsilon}_{n-2} \mid \bar{\varepsilon}_1 \cap \bar{\varepsilon}_2 \cdots \cap \bar{\varepsilon}_{n-3}]$$

$$\geq \left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\cdots\left(1-\frac{2}{n-i+1}\right)\cdots\left(1-\frac{2}{3}\right)$$

$$= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right)\left(\frac{n-5}{n-3}\right)\cdots\left(\frac{2}{4}\right)\left(\frac{1}{3}\right)$$

$$= \frac{2}{n(n-1)} = \binom{n}{2}^{-1}$$

Multiple Runs of Contraction Algorithm

- With $\binom{n}{2}$ runs we get:

$$\Pr[\text{failure}] \leq \left(1-\frac{1}{\binom{n}{2}}\right)^{\binom{n}{2}} \leq \frac{1}{e} \approx 36\%$$

- With $\binom{n}{2} \ln n$ runs:

$$\Pr[\text{failure}] \leq \left(\frac{1}{e}\right)^{\ln n} = \frac{1}{n}$$

Hashing (good: compact/efficient, minimizes collisions) | U mod n | U mod p – p is prime | closest on

a function that converts $\mathbb{Z}$ I into hash val.

(I) o Input: large universe of values U,

$$|U| \gg n$$

o Output: a hash value for $u \in U$ to $\{0, 1, \ldots n-1\}$

– Dictionary Data Structure
→ Storage of a subset of values from U
→ a map where the key is generated hashed efficiently from the value

Operations
Make Dictionary — Initializes a fresh dictionary that can maintain a subset S of U (initially ∅)
– Insert(u) adds $u \in U$ to dictionary (S)
– Delete(u) Remove u from S
– Lookup(u) Determine if $u \in S$ if so retrieve it

Motivation
– The values in U may be huge (keywords)
– value $u \in U$ is used to build smaller keys

Hashing
Hashing Table: a n-length array H to store the values
Hash Function: Map $u \in U$ to an index in H:
$h: U \to \{0, \ldots, n-1\}$

Dictionary Hashing
Collision: $h(u) = h(v)$
where H[i] becomes a bucket (linked list) to store values $h(u) = i$

Lemma 2
Given h(x) the prob. $h(u) = h(v)$ for $u, v \in U$ is $1/n$

Need: $\Pr[h(v) = x \mid h(u) = x] = 1/n$

Recall SAT

$(x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (x_2 \vee \bar{x}_3)$

- ✓ boolean terms $x_1, \ldots, x_n$
- ✓ $\bar{x}_i \leftrightarrow \neg x_i$
- ✓ C_j - clause is disjunct of distinct terms $(x_1 \vee \bar{x}_2)$
- ✓ length of C_j is $|C_j|$
- ✓ conjuction of k clauses $C_1 \wedge C_2 \wedge \ldots \wedge C_k$
- ✓ Truth assignment $v: X \to \{0,1\}$
- ✓ v is a satisfying assignment if C_i evaluate to 1

MAX 3-SAT

Given a set of literals $X: x_1, \ldots, x_n$ and a collection of clauses $C: C_1 \wedge C_2 \wedge \ldots \wedge C_k$ each length of 3, does there exist a satisfying assignment?

MAX 3-SAT Problem → given 3SAT, find as many clause possible

Random Assignment

For each x_i independently assign a value of 0 or 1 with probability 1/2 each

$\Pr[Z_i = 0] = \left(\frac{1}{2}\right)^3 = \frac{1}{8}$

$\mathbb{E}[Z_i] = 7/8 \Rightarrow 1 \cdot 7/8 + 0 \cdot 1/8$

since $\Pr[Z_i = 1] = 1 - \Pr[Z_i = 0] = 7/8$

Clause C_j

Let Z_i be a random var. 1 if satisfied otherwise

Let $Z = \sum_{i=1}^{k} Z_i$

$\mathbb{E}[Z] = \mathbb{E}\left[\sum_{i=1}^{k} Z_i\right]$

$= \mathbb{E}[Z_1] + \mathbb{E}[Z_2] + \ldots + \mathbb{E}[Z_k]$

$= 7/8\, k$

Theorem 8

✓ Let p_j be the probability that j clauses are satisfied

✓ we want to calculate $p = \sum_{j \geq 7/8 k} p_j$

✓ $7/8\,k = \sum_{j=0}^{k} j p_j = \left[\sum_{j < 7/8k} j p_j + \sum_{j \geq 7/8k} j p_j \right]$

$\leq \left(\frac{7k}{8} - \frac{1}{8} \right) \sum_{j < 7/8k} p_j + k \sum_{j \geq 7/8k} p_j$

$\left(\frac{7}{8} - \frac{1}{8} \right)(1-p) + kp \leq \left(\frac{7k}{8} - \frac{1}{8} \right) + kp$

$p \geq \frac{7/8\,k - \left(\frac{7k}{8} - \frac{1}{8} \right)}{k} = \frac{1}{8k}$ □

With $p = 1/8k$ we have Bernoulli trials

I.e. The expected runtime is $8k$ runs of random assignments