Maximum Subarray Problem

January-08-13 2:51 PM

Problem Solving

Problem → prove lower bounds → may need to change problem if too hard
↓
Design an algorithm
↓ ↑ do better
Analyze algorithm
↓
Program

Problem (Bentley)

Given numbers $a_1, \dots, a_n$ find a sub-block (contiguous subsequence) $a_i, a_{i+1}, \dots, a_j$ with maximal sum.

Example

```
Input: 1, -6, 3, -1, 4, 2, -3, 2
Output: 8
```

Algorithm 0

Brute force

```
1) ans = 0
2) for i = 1 to n do
3)     for j = i to n do
4)         sum = 0
5)         for k = i to j do
6)             sum += a_k
7)         if sum > ans then ans = sum
8) return ans
```

Analysis

Lines 5-6: $c(j-i+1) \leq cn$ for some constant c. c depends on compiler/ machine
Lines 3-7: $\leq n(cn + c')$ for some constant c' depending upon compiler or machine
$= n \cdot O(n) = O(n^2)$
Total time: $n \cdot O(n^2) = O(n^3)$
In fact, is $\Theta(n^3)$

e.g. $n = 10^6,\ n^3 = 10^{18},\ 10^9$ ops/sec $\Rightarrow 10^9$sec ~ 30 years

Algorithm 1

Idea - don't recompute sum from scratch

```
ans = 0
for i = 1 to n do
    sum = 0
    for j = i to n do
        sum += a_j
        if sum > ans then ans = sum
return ans
```

Analysis

$O(n^2)$

Algorithm 2

Idea - divide and conquer

```
solve(a_1, ..., a_n)
if n = 1 then return max(a_1, 0)
ans = max{solve(a_1, ..., a_⌊n/2⌋), solve(a_⌊n/2⌋+1, ..., a_n)}
ansl = sum = 0
for i = ⌊n/2⌋ to 1 do
    sum = sum + a_i
    if sum > ansl then ansl = sum
ansr = sum = 0
for i = ⌊n/2⌋ + 1 to n do
    sum = sum + a_i
    if sum > ansr then ansr = sum
return max(ans, ansl + ansr)
```

Analysis

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2T\left(\frac{n}{2}\right) + O(n) & \text{otherwise} \end{cases}$$

$O(n \log n)$

Algorithm 3

Dynamic programming

Let b_j be the maximum sum over all blocks that end at j

$b_j = \max(b_{j-1} + a_j, a_j)$

```
ANS = 0
b = 0
FOR j = 1 .. n
    b = max{b + a_j, a_j}
    ANS = max{b, ANS}
}
RETURN ANS
```

$O(n)$ runtime

Any algorithm must take $\Omega(n)$, so it is $\Theta(n)$

3SUM Problem

January-10-13 2:56 PM

3SUM Problem

Given n numbers $a_1, \ldots, a_n$ and a target t does there exist $i, j, k \in \{1 \ldots n\}$ such that $a_i + a_j + a_k = t$? (not necessarily distinct)

Example

30, 12, 8, 37, 33, 82 $\quad t = 50$

$30 + 12 + 8 = 50 \Rightarrow i = 1, j = 2, k = 3$

Algorithm 1

```
for i = 1 to n
    for j = 1 to n
        for k = 1 to n
            if aᵢ + aⱼ + aₖ = t then
                return Yes
return No
```

Runtime $O(n^3)$

Algorithm 2

Let $A = \{a_i + a_j : i, j \in \{1, \ldots, n\}\}$

Let $B = \{t - a_k : k \in \{1, \ldots, n\}\}$

If A and B have a common element return Yes, else return No

Check by soring A and B individually, then scanning through each simultaneously.

(In class sorted $A \cup B$ and checked for pair of elements each in a different set, but this is simpler)

Make A: $O(n^2)$

Make B: $O(n)$

Sort A: $O(n^2 \log n^2) = O(2n^2 \log n) = O(n^2 \log n)$

Sort B: $O(n \log n)$

Scan: $O(n^2)$

Total Runtime: $O(n^2 \log n)$

Algorithm 3

$A_i = \{a_i + a_j : j \in \{1, \ldots, n\}\}$

$B = \{t - a_k : k \in \{1, \ldots, n\}\}$

Pre-sort input. Get A_i and B sorted automatically.

```
for i = 1 to n
    if Aᵢ and B have a common element
        return Yes
return No
```

Pre-sort input: $O(n \log n)$

Check if A_i and B have a common element: $O(n)$

Repeated n times $\Rightarrow O(n^2)$ runtime

Full In-Class Algorithm

```
Sort input array a
Generate B:
    Bᵢ = t - aₙ₋ᵢ for i = 1 to n
for i = 1 to n
    Aᵢ  = { aᵢ + aⱼ | j = 1 to n }
    Merge sorted Aᵢ and B to obtain Aᵢ ∪ B (also sorted)
    If Aᵢ and B have a common element
        return Yes
return no
```

Asymptotic Notation

January-15-13 2:56 PM

Common Summations

$$\sum_{i=1}^{n} i^d = \Theta(n^{d+1}) \ \forall\, d \geq 0$$

$$\sum_{i=1}^{n} c^i = \frac{c^{n+1} - c}{c - 1} = \begin{cases} \Theta(1) & \text{for } c < 1 \\ \Theta(n) & \text{for } c = 1 \\ \Theta(c^n) & \text{for } c > 1 \end{cases}$$

Harmonic Series

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

Stirling's formula:

$$\log(n!) = \sum_{\mathrm{i}=1}^{\mathrm{n}} \log i = n \log n - \Theta(n)$$

Connection with Limits

1. $f(n) \in o(g(n))$ iff
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$
2. $f(n) \in \omega(g(n))$ iff
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$
3. $f(n) \in O(g(n))$ iff
$$\limsup_{n\to\infty} \frac{f(n)}{g(n)} = \text{const} < \infty$$
4. $f(n) \in \Omega(g(n))$ iff
$$\liminf_{n\to\infty} \frac{f(n)}{g(n)} = \text{const} > 0 \text{ or } \infty$$

Example

```
for i = 1 to n do
    for j = 1 to n do             Θ(n)
    for l = 1 to i do             Θ(i²)
        for k = 1 to i do         Θ(i)
```

Total:

$$\sum_{i=1}^{n} (n + i^2) = \Theta(n^2) + \sum_{i=1}^{n} i^2 = \Theta(n^2) + \Theta(n^3) = \Theta(n^3)$$

Proof of Harmonic Series Sum Value

$$\sum_{i=1}^{n} \frac{1}{i} \leq \int_1^n \frac{1}{x} dx + 1 = \ln n + 1$$

Build-Heap Example

```
for i = n down to 1 do
    j = 1
    while j ≤ n do
        j = 2j
```

t steps total. What is t?

$2^{t-1} i < n < 2^t i$

$2^{t-1} < \frac{n}{i} < 2^t$

Total $O(n \log n)$ (Stirling's formula)

Tight bound

$$\Theta\left(\sum_{i=1}^{n} \log \frac{n}{i}\right) = \Theta\left(\sum_{i=1}^{n} (\log n - \log i)\right) = \Theta\left(\sum_{i=1}^{n} \log n - \sum_{i=1}^{n} \log i\right) = \Theta\left(n \log n - \left(n \log n - \Theta(n)\right)\right)$$
$$= \Theta(n)$$

Solving Recurrences

January-17-13 2:47 PM

Thee methods of solving recurrences
1. Recursion Tree Method
2. Master Method
3. Guess-and-check

Recursion Tree Method

- Expand for k iterations to get a tree of terms
- Set k to reach the base case
- Sum across rows, levels

Master Theorem

Let

$$T(n) = \begin{cases} aT\left(\frac{n}{b}\right) + f(n) & \text{if } n \geq n_0 \\ c & \text{otherwise} \end{cases}$$

Set $d = \log_b a$ and pick $\epsilon > 0$

Case 1: $f(n) = O\left(n^{d-\epsilon}\right) \Rightarrow T(n) = \Theta\left(n^d\right)$

Case 2: $f(n) = \Theta\left(n^d\right) \Rightarrow T(n) = \Theta\left(n^d \log n\right)$

Case 3: $\frac{f(n)}{n^{d+\epsilon}}$ is increasing $\Rightarrow T(n) = \Theta\big(f(n)\big)$

$f(n) = \Omega\left(n^{d+\epsilon}\right)$

Guess and Check Method

Exactly what it sounds like.
- Guess the solution $T(n) \leq cn^k$
- Verify by inductive proof
- Fill in constants

Example Recurrence Relation

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + n & \text{for } n > 1 \\ 1 & \text{if } n = 1 \end{cases}$$

Example of Master Method

Merge sort:

$a = 2, b = 2, f(n) = n$
$d = \log_2 2 = 1$
Case 2 so $T(n) = \Theta(n \log n)$

Example 2

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + n^2 & \text{for } n > 1 \\ 1 & \text{if } n = 1 \end{cases}$$

$a = b = 2, \quad f(n) = n^2$
$d = 1, \quad \epsilon = 0.01$
$\frac{n^2}{n^{1.01}}$ is increasing
Case 3 so $T(n) = \Theta(n^2)$

Example 3

$$T(n) = \begin{cases} 5T\left(\frac{n}{4}\right) + \sqrt{n} & \text{if } n > 1 \\ 1 & \text{if } n = 1 \end{cases}$$

$d = \log_4 5 \approx 1.161, \quad f(n) = n^{0.5}$
$n^{0.5} = O\left(n^{\log_4 5 - \epsilon}\right)$
Case 1 so $T(n) = \Theta\left(n^{\log_4 5}\right)$

Example 4

$T(n) = T\left(\frac{n}{2}\right) + 1$
$a = 1, \quad b = 2, \quad f(n) = 1$
$d = \log_2 1 = 0$
Case 2 so $T(n) = \Theta(\log n)$

Example 5

$T(n) = 2T\left(\frac{n}{4}\right) + n$

$a = 2, \quad b = 4, \quad d = \log_4 2 = \frac{1}{2}$

$\frac{n}{n^{\frac{1}{2}+\epsilon}}$ is increasing
Case 3 $T(n) = \Theta(n)$

Example 6

$T(n) = 2T\left(\frac{n}{2}\right) + \frac{n}{\log n}$

$d = 1$
Case 1: No
Case 2: No
Case 3: No

The master theorem does not apply.
By recursion tree method:

$$T(n) = \frac{n}{\log n} + 2\frac{\frac{n}{2}}{\log\left(\frac{n}{2}\right)} + 4\frac{\left(\frac{n}{4}\right)}{\log\left(\frac{n}{4}\right)} + \cdots$$

$$= \sum_{k=0}^{\log n - 1} 2^k \frac{\frac{n}{2^k}}{\log n - k} = n \sum_{k=0}^{\log n - 1} \frac{1}{\log n - k} = n \sum_{i=1}^{\log n} \frac{1}{i} = n \times \Theta(\log \log n)$$

$= \Theta(n \log \log n)$

Example Guess and Check

Example 1

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + n^2 & \text{for } n \geq 1 \\ 1 & \text{if } n = 1 \end{cases}$$

Guess $T(n) \leq cn^2$

Base case: $n = 1$
$T(n) = 1 \leq cn^2$ for $c \geq 1$
Inductive step:

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2 \leq \frac{2cn^2}{4} + n^2 = \left(\frac{c}{2} + 1\right)n^2 \leq cn^2$$

$$\Rightarrow \left(\frac{c}{2} + 1\right) \leq c \Rightarrow 1 \leq \frac{1}{2}c \Rightarrow c \geq 2$$

Pick $c = 2$
Know $T(n) \in O(n^2)$
$T(n) = 2T\left(\frac{n}{2}\right) + n^2 \geq n^2 \Rightarrow T(n) \in \Omega(\mathrm{n}^2)$
$\therefore T(n) = \Theta(n^2)$

Example 2

$$T(n) = \begin{cases} 3T\left(\frac{n}{2}\right) + 4T\left(\frac{n}{4}\right) + 1 & \text{for } n > 1 \\ 1 & \text{for } n = 1 \end{cases}$$

Guess $T(n) \leq cn^2$
Base case: $T(1) = 1 \leq cn^2$ for $c \geq 1$
Inductive step:

$$T(n) = 3 \cdot c\left(\frac{n}{2}\right)^2 + 4c\left(\frac{n}{4}\right)^2 + 1 = \frac{3}{4}cn^2 + \frac{1}{4}cn^2 = cn^2 + 1 \nleq cn^2$$

Problem. Instead assume
$T(n) \leq cn^2 - c'$

Base case: $T(1) = 1 \leq cn^2 - c'$ for $c - c' \geq 1$
Inductive step:
$T(n) = 3 \cdot c\left(\frac{n}{2}\right)^2 - 3 + 4 \cdot c\left(\frac{n}{4}\right)^2 - 4 + 1 = cn^2 - 6 \leq cn^2 \;\; \forall c > 0$
Set $c = 2, c' = 1$. So
$T(n) \leq 2n^2 - 1 \Rightarrow T(n) \in O(n^2)$

Divide and Conquer

January-17-13 3:45 PM

Divide and Conquer

- Divide into subproblems
- Recurse
- Combine solutions

Domination

A point p dominates point q iff $p_i \geq q_i \; \forall i$
A point q is maximal for a set S if no point in a set S dominates it.

Maximal Problem

Find all maximal points in the set S.

Brute-force algorithm

For each $q \in P$ count # of points $p \in P$ that dominate q. If no points domination q then return it.
$\Theta(n^2)$

Divide and Conquer

```
Maximal(p_1, ..., p_n)
Input p_1, ..., p_n a list of 2D points pre-sorted by x coordinate
If n = 1 then return {p_1}
{q_1, ..., q_l} = Maximal(p_1, ..., p_⌊n/2⌋)
{s_1, ..., s_m} = Maximal(p_⌊n/2⌋+1, ..., p_n)
i = 1
while i ≤ l and q_i.y > s_1.y do
    i = i + 1
return {q_1, ..., q_{i-1}, s_1, ..., s_m}
```

Analysis

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + \Theta(n) & \text{if } n > 1 \\ 1 & \text{otherwise} \end{cases}$$

$T(n) = \Theta(n \log n)$

Have initial sorting as well that takes time $\Theta(n \log n)$

Notes

1) There is a $\Omega(n \log n)$ bound for comparison-based algorithms
2) Can get $\Theta(n \log n)$ without divide and conquer (DAC)
3) DAC solves more general "dominance counting" problems

If the points are already sorted then we can solve this in $\Theta(n)$ time and dominance counting in $O\left(n\sqrt{\log n}\right)$ time (2010)

Union of Intervals Problem

Given n intervals $[s_1, t_1], [s_2, t_2], \ldots, [s_n, t_n]$
Compute their union
Output sorted list $c_1, c_2, \ldots, c_{2k}$
representing intervals $[c_1, c_2], [c_3, c_4], \ldots, [c_{2k-1}, c_{2k}]$

Note that
$A_1 \cup A_2 \cup \cdots \cup A_n = \left(A_1 \cup \cdots \cup A_{\lfloor \frac{n}{2} \rfloor}\right) \cup \left(A_{\lfloor \frac{n}{2} \rfloor + 1} \cup \cdots \cup A_n\right)$

So

```
Union([s_1, t_1], [s_2, t_2], ..., [s_n, t_n]) :
    if n == 1 then return (s_1, t_2)
    (a_1, ..., a_l) = Union([s_1, t_1], ..., [s_⌊n/2⌋, t_⌊n/2⌋])
```

```
    (b_1, ..., b_m) = Union([s_{⌊n/2⌋+1}, t_{⌊n/2⌋+1}], ..., [s_n, t_n])
    return Merge_Intervals(a, b)

Merge_Intervals a, b:
i = 1, j = 1
do {
    if a_i < b_j then
        if j is odd then {k++, c_k = a_i}
        i++
    else
        if i is odd then {k++, c_k = b_j}
        j++
} while i != len(a) || j != len(b)
return (c_1, ..., c_k)
```

Observations

1. Elements are considered for inclusion in sorted order
2. If $a_i < b_j$ and a_i and b_j are being compared, we must have had $b_{j-1} \leq a_i$

Analysis

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + \Theta(n) & \text{if } n > 1 \\ 1 & \text{otherwise} \end{cases}$$

$$T(n) = \Theta(n \log n)$$

Closest Pair

January-24-13 2:31 PM

Closest Pair Problem

Given a set P of points in 2d find a pair (p, q) with the smallest distance
$d(p, q) = \sqrt{(p.x - q.x)^2 + (p.y - q.y)^2}$

Brute-force approach: $\Theta(n^2)$ time

First idea: Divide vertically
Get minimum distance for left and right sides. Then only have to look with in min distance of the vertical dividing line for other pairs.

Shamos Algorithm

```
ClosestPair(P)
    if n ≤ 3
    x_m = median x-coordinate of P                          Θ(n)
    P_L = {p ∈ P : p.x ≤ x_m}
    P_R = {p ∈ P : p.x > x_m}                               Θ(n)
    δ_L = ClosestPair(P_L)
    δ_R = ClosestPair(P_R)
    δ = min{δ_L, δ_R}
    ⟨p_1, …, p_L⟩ = SORT({p ∈ P: x_m − δ ≤ p.x ≤ x_m + δ})  Θ(n) +
    O(n log n)
    for i = 1 to L                                          O(n)
        for j = i+1, i+2, … as long as p_j.y ≤ p_i.y + δ    Θ(1)
            δ = min{δ, d(p_i, p_j)}
```

Observation 1

Only need to look at points within δ of x_m

Observation 2

Only need to compare points such that $|p.y - q.y| \le \delta$
Solution must be within some $\delta \times 2\delta$ rectangle

Observation 3

We can bound the number of points in each such rectangle. There are at most 6 points.

Analysis

$T(n) = 2T\left(\frac{n}{2}\right) + \mathrm{O}(n \log n)$

$T(n) = 2T\left(\frac{n}{2}\right) + \Omega(n)$

Tree method:

$T(n) = 2T\left(\frac{n}{2}\right) + O(n \log n) \Rightarrow T(n) = \mathrm{O}(n \log^2 n)$

$T(n) = 2T\left(\frac{n}{2}\right) + \Omega(n) \Rightarrow T(n) = \Omega(n \log n)$

Improvement: Pre-sort the points according to x and y coordinates.
No need to sort on each recursive step. $T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) = \Theta(n \log n)$

Large Multiplication

January-29-13 3:09 PM

Large Integer Multiplication

Given to n-bit numbers $A = a_{n-1}a_{n-2}\dots a_0$

$B = b_{n-1}b_{n-1}\dots b_0$

Compute $AB = c_{2n-1}c_{2n-2}\dots c_0$

Algorithm

Idea: Divide A and B into blocks of $\frac{n}{2}$ digits

$$A = a_{n-1}a_{n-2}\dots a_{\frac{n}{2}}|a_{\frac{n}{2}-1}a_{\frac{n}{2}-2}\dots a_0 = A'\cdot 2^{\frac{n}{2}} + A''$$

$$B = b_{n-1}b_{n-2}\dots b_{\frac{n}{2}}|b_{\frac{n}{2}-1}b_{\frac{n}{2}-2}\dots b_0 = B'\cdot 2^{\frac{n}{2}} + B''$$

$$AB = \left(A'\cdot 2^{\frac{n}{2}} + A''\right)\left(B'\cdot 2^{\frac{n}{2}} + B''\right) = A'B'\cdot 2^n + (A'B'' + A''B')\cdot 2^{\frac{n}{2}} + A''B''$$

$$T(n) = 4\,T\left(\frac{n}{2}\right) + \Theta(n) = \Theta(n^2)$$

No improvement

A clever idea:

$A'B'' + A''B' = (A' + A'')(B' + B'') - A'B' - A''B''$

Now have 3 multiplications, 6 additions

```
Mult(A, B)
    divide A into A', A''
    divide B into B', B''
    C_1 := Mult(A', B')
    C_2 := Mult(A'', B'')
    C_3 := Mult(A'+A'', B'+B'')
    return C_1 · 2^n + (C_3 − C_2 − C_1) · 2^(n/2) + C_2
```

Analysis

$$T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n) = \Theta\left(n^{\log_2 3}\right) \approx \Theta(n^{1.59})$$

Refinements

3-way divide

Can do with 5 multiplications, messy formula

$$T(n) = 5T\left(\frac{n}{3}\right) + \Theta(n) = \Theta\left(n^{\log_3 5}\right) \approx \theta(n^{1.47})$$

4-way divide

Can do with 7 multiplications

$$T(n) = 7T\left(\frac{n}{4}\right) + \Theta(n) = \Theta\left(n^{\log_4 7}\right) = \Theta(n^{1.41})$$

In fact, we can get $O\left(n^{1+\delta}\right)$ for any δ by dividing enough times.

Current best algorithm:

Schontage, Strassen (1971) $O(n\log n\log\log n)$

Furer (2007) $O\left(n\log n\, 2^{\log^* n}\right)$

$\log^* n$

is the number of times you need to apply $\log_2 \square$ to n to get 2.

If $2 = \log\log\log n$ then $\log^* n = 3$

Large Matrix Multiplication

Given 2 matrices $A, B \in \mathbb{R}^{n\times n}$
Compute the $n \times n$ matrix $C = AB$

Standard Method

```
for i = 1 to n
    for j = 1 to n
```
$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

$\Theta(n^3)$ time

Strassen Algorithm (1969)

Partition:

$$A = \begin{bmatrix} A_1 & | & A_2 \\ - & + & - \\ A_3 & | & A_4 \end{bmatrix}, \qquad B = \begin{bmatrix} B_1 & | & B_2 \\ - & + & - \\ B_3 & | & B_4 \end{bmatrix}$$

$$AB = \begin{bmatrix} A_1B_1 + A_2B_3 & | & A_1B_2 + A_2B_4 \\ - & + & - \\ A_3B_1 + A_4B_3 & | & A_3B_2 + A_4B_4 \end{bmatrix}$$

8 additions, 4 multiplications

$$T(n) = 8\,T\left(\frac{n}{2}\right) + \Theta(n^2) = \Theta(n^3)$$

A clever idea:

$C_1 = A_1(B_1 - B_3)$
$C_2 = (A_1 + A_2)B_3$
$C_3 = A(B_3 - B_2)$
$C_4 = (A_3 + A_4)B_2$
$C_5 = (A_1 + A_4)(B_2 + B_3)$
$C_6 = (A_2 - A_4)(B_3 + B_4)$
$C_7 = (A_3 - A_1)(B_1 + B_2)$

$$AB = \begin{bmatrix} C_1 + C_2 & | & C_5 + C_6 + C_3 - C_2 \\ - & + & - \\ C_5 + C_7 + C_1 - C_4 & | & C_3 + C_4 \end{bmatrix}$$

$$T(n) = 7\,T\left(\frac{n}{2}\right) + \Theta(n^2) = \Theta\left(n^{\log_2 7}\right) = \Theta(n^{2.81})$$

Note

This algorithm is not often used for floating point matrices because it accumulates errors quickly.
It is good for large integer multiplication.

It has applications to other operations (e.g. $Ax = b$, inverse, determinant) and to graph problems

Another algorithm:
Pan(1798)

$$T(n) = 143640\,T\left(\frac{n}{70}\right) + \Theta(n^2) \Rightarrow O(n^{2.796})$$

Best so far:
Vassilevska- Williams
$O(n^{2.373})$

Median Algorithm

February-05-13 2:32 PM

Given n numbers $a_1, \ldots, a_n$ (unsorted) and k, find the k^{th} smallest element.

Algorithm 1

Sort and choose k^{th} $\Theta(n \log n)$

Algorithm 2

Remove largest element k times $\Theta(kn)$

Algorithm 3

Make a heap and remove the k largest elements $\Theta(n + k \log n)$

Algorithm 4

Quick-select (see CS 240)
Choose pivot and recurse on appropriate half.

Works well with random pivot.
Deterministic version:
Break the numbers into groups of 5. Choose the pivot the be the median of the medians of those groups of 5.

Claim

If i is the index of the median of medians x then

$$\frac{3n}{10} \leq i \leq \frac{7n}{10}$$

$\sim \dfrac{n}{10}$ groups G_i such that $x_i \leq x$

and each such group contains 3 numbers $\Rightarrow$ at least $\frac{3n}{10}$ numbers are $\leq n$

Similarly, $\frac{7n}{10}$ numbers are $\geq n$

$$T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + \Theta(n)$$

$$T(n) = n \sum_{i=1}^{\log n - 1} \left(\frac{9}{10}\right)^i = \Theta(n)$$

Greedy Algorithms

February-05-13 3:17 PM

Greedy Algorithms

- Incrementally build solution
- At each step, choose what seems to the best at the moment
- Optimization problems (find solution maximizing or minimizing some function)
- Advantages: simple and fast
- Disadvantages: may not be correct

Example: Coin Changing

Find minimum number of coins adding up to W
With Canadian currency can repeatedly pick largest possible coin.

Example

Given n intervals $[a_1, b_1], [a_2, b_2], \dots, [a_n, b_n]$

find the maximum number of disjoint intervals.

Idea 1:

Pick interval of minimum width. Does not work
Counterexample:

```
______  ______
    ____
```

Idea 2:

Pick interval that conflicts with the fewest other intervals. Also does not work.
Counterexample:

```
  ____       ____
  ____  ____  ____
____  ____  ____  ____
____              ____
```

Least number of conflicts is the middle interval in 2nd row when 3rd row is optimal.

Greedy Algorithm

```
repeat {
    choose interval [a, b] with smallest b
    remove [a, b] & all  intervals intersecting it
}
```

Implementation

Naïve: $O(n^2)$
Sort and scan $O(n \log n)$

Correctness Proof

To show there exists an optimal solution consisting of all the chosen intervals.
Let I^* be any optimal solution.
Let $[a^*, b^*]$ be the leftmost interval in I^*
Let $[a, b]$ be the leftmost interval chosen by our algorithm.
We can safely swap $[a^*, b^*]$ with $[a, b]$
Then $(I^* - \{[a^*, b^*]\}) \cup [a, b]$ would be a valid solution with the same number of intervals.
Repeat argument ignoring $[a^*, b^*]$ and $[a, b]$. ■

More Greedy Algorithms

February-07-13 2:51 PM

Fractional Knapsack Problem

Given "values" $v_1, \dots, v_n > 0$ and "weights" $w_1, \dots, w_n > 0$
Capacity $W > 0$

Maximize $\sum_{i=1}^{n} v_i x_i$ such that $\sum_{i=1}^{n} w_i x_i \le W$
over $0 \le x_1, \dots, x_n \le 1$

Greedy Solution

Repeatedly pick as much as possible of item i such that $\frac{v_i}{w_i}$ is maximized over remaining items.

Correctness

To show ∃optimal solution $(x_1^*, x_2^*, \dots, x_n^*)$ with $x_j^* = x_j$ for all iterations
Proof: (Assume ratios are distinct)
Consider the first iteration.
Suppose $x_j^* \ne x_j$
Know $x_j^* \le x_j \Rightarrow x_j^* < x_j$
Find another item k with $x_k^* > 0$
Increase x_j^* by $\frac{\delta}{w_j}$, decrease x_k^* by $\frac{\delta}{w_k}$ where $\delta > 0$ is sufficiently small
$\sum_{i=1}^{n} w_i x_i^*$ is unchanged but
$\sum_{i=1}^{n} v_i x_i^*$ increases by $\left(v_j \frac{\delta}{w_j} - v_k \frac{\delta}{w_k}\right) = \left(\frac{v_j}{w_j} - \frac{v_k}{w_k}\right)\delta > 0$
⇒ Contradiction
Repeat argument for other iterations

Stable Marriage Problem

Given n candidates and n employers,
and a preference list (a permutation of employers) for each candidate
and a preference list (a permutation of candidates) for each employer
We can't have both C prefers E' over E and E' prefers C over C'

Example

$C_1: E_4, E_3, E_1, E_2$
$C_2: E_1, E_3, E_2, E_4$
$C_3: E_4, E_3, E_2, E_1$
$C_4: E_3, E_1, E_4, E_2$
$E_1: C_1, C_2, C_3, C_4$
$E_2: C_1, C_3, C_2, C_4$
$E_3: C_3, C_2, C_4, C_1$
$E_4: C_2, C_3, C_1, C_4$

Bad Idea:

Brute force $O(n!)$

The "Natural" Algorithm

1. Start with any matching
2. While ∃ to unstable pairs (C, E) and (C', E') replace with new matchings (C, E') and (C', E)
   Can also be slow (may not terminate)

The "Real Life" Greedy Algorithm (Gabe, Shapley 1962)

1. While ∃ unmatched employer E do {
   a. pick $C =$ next best candidate in E's preference list
      ("next" means candidate which has not yet been contacted by E)
      // E makes an offer to C
   b. If C is unmatched or C prefers E over C's current employer E_0 then unmatch (C, E_0) and match (C, E)

Run on example:

$E_1: C_1$
$E_2: \cancel{C_1}\, \cancel{C_3}\, \cancel{C_2}\, C_4$
$E_3: \cancel{C_1}\, C_2$
$E_4: \cancel{C_2}\, C_3$

Solution
$(E_1, C_1), (E_2, C_4), (E_3, C_2), (E_4, C_3)$

Analysis

Each employer makes $\le n$ offers so n^2 iterations
Spend n^2 time to make a table of $P[i, j] =$ position of E_i in C_i 's preference list
So b takes $O(1)$ time
∴ Whole algorithm runs in $O(n^2)$ time.

Claim: in b), C exists.
Proof: Some candidate is unmatched because there are n possible candidates and at most $n - 1$ matches.

Proof of Correctness

Matching is stable

Proof by contradiction:

Suppose we have matched pairs $(C, E), (C', E')$ $where$

i) C prefers E' over E

ii) E' prefers C over C'

ii $\Rightarrow$ E' makes offer to C before C'

$\Rightarrow$ both E and E' have made an offer to C

$\Rightarrow$ C prefers E over E'

$\Rightarrow$ Contradiction with i

Dynamic Programming

February-12-13 3:16 PM

Example: Binomial Coefficients

$\binom{n}{k}$ = # of size k subsets of $\{1, \dots, n\}$

$$\binom{n}{k} = \begin{cases} 1 & \text{if } k = 0 \text{ or } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{otherwise} \end{cases}$$

Divide and conquer algorithm:
$T(n) \le 2T(n-1) + O(1) \Rightarrow T(n) = O(2^n)$

Instead, fill record temporary values in a table

n \ k	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4	1	4	6	4	1	
5	1	5	10	10	5	1

$O(n^2)$ time
Can be more careful and do it in $\Theta(nk)$ time

Largest Common Subsequence (LCS)

Given 2 strings $A = a_1 a_2 a_3 \dots a_n$ and $B = b_1 b_2 b_3 \dots b_m,\ m < n$
Find the maximum string $c_1 \dots c_k$ such that
$c_1 = a_{i_1} = b_{j_1}$
$\vdots$
$c_k = a_{i_k} = b_{j_k}$
where
$i_1 < i_2 < \cdots < i_k$
$j_1 < j_2 < \cdots < j_k$

Brute-force: Check all subsequences in A, B $\Rightarrow O(2^{n+m})$

Dynamic Programming

Let $C[n, m]$ be the length of $LCS(a_1 \dots a_n, b_1 \dots b_m)$
Base case: $C[i, 0] = C[0, j] = 0$ for $i = 0 \dots n, j = 0 \dots m$

If $a_i = b_j$ then
$\quad C[i,j] = C[i-1, j-1] + 1$
Otherwise
$\quad C[i,j] = \max(C[i-1,j], C[i, j-1])$

Base cases:
$C[i, 0] = 0, \qquad C[j, 0] = 0$

Example

			L	O	G	A	R
	i \ j	**0**	**1**	**2**	**3**	**4**	**5**
	0	0	0	0	0	0	0
A	**1**	0	0	0	0	1	1
L	**2**	0	1	1	1	1	1
G	**3**	0	1	1	2	2	2
O	**4**	0	1	2	2	2	2
R	**5**	0	1	2	2	2	3

More Problems

February-26-13 2:33 PM

0/1 Knapsack problem

Given values $v_1, \dots, v_n > 0$, and weights $w_1, \dots, w_n > 0$
capacity W (integers)

Find subset $S \subseteq \{1, .., n\}$ maximizing $\sum_{k \in S} v_k$ such that $\sum_{k \in S} w_k \leq W$
Define a sub-problem

$$c[i,j] = \max \sum_{k \in S} v_k \text{ s.t. } \sum_{k \in S} w_k \leq j \text{ over all } S \subseteq \{1, \dots, i\}$$

Find answer: $C[n, W]$
Base cases:

$C[i, 0] = 0 \; \forall i$
$C[0, j] = 0 \; \forall j$

Recurrence:

$$C[i,j] = \begin{cases} \max\left(C[i-1,j], C[i-1, j-w_j] + v_i\right) & \text{if } w_j < j \\ C[i-1,j] & \text{otherwise} \end{cases}$$

Analysis

$O(nW)$, but W may be arbitrarily large.

Can we have a polynomial algorithm in n?

Edit Distance

Goal: Minimize the number of operations that transform string $A = a_1 \dots a_n$ into $B = b_1 \dots b_m$
Edit operations:

- substitution
- insertion
- deleting

Application

Compare sequences of DNA

1. How many mutations between Human and Chimp DNA? ⇒ Edit distance
2. Which bases in humans correspond to which bases in chimps? ⇒ Sequence alignment

Needleman-Wunsch Algorithm

Define $C[i,j]$ = edit distance between $a_1 \dots a_i$ and $b_1 \dots b_j$
$\pi[i,j] \in \{\text{Match}, \text{Deletion}, \text{Insertion}\}$ ⇒state of last (column?) in alignment

Base case:

$C[i, 0] = i$
$C[0, j] = j$

Recurrence

$$C[i,j] = \begin{cases} \min(C[i,j-1]+1, C[i-1,j]+1, C[i-1,j-1]+1) & \text{if } a_i \neq b_j \\ \min(C[i,j-1]+1, C[i-1,j]+1, C[i-1,j-1]) = C[i-1,j-1] & \text{if } a_i = b_j \end{cases}$$

Analysis

$O(nm)$ time
$O(nm)$ space

If you do not need alignment information, can do in $O(\min(n,m))$ space.
Can be careful and get alignment in $O(\min(n,m))$ space

- Hirschberg's trick

Example

BANANA & PANDA
C

			P	A	N	D	A
	i \ j	0	1	2	3	4	5
	0	0	1	2	3	4	5
B	1	1	1	2	3	4	5
A	2	2	2	1	2	3	4
N	3	3	3	2	1	2	3
A	4	4	4	3	2	2	3
N	5	5	5	4	3	3	3
A	6	6	6	5	4	4	3

Π

			P	A	N	D	A
	i \ j	0	1	2	3	4	5
	0						
B	1		M	I	I	I	I
A	2		M	M	I	I	I
N	3		M	D	M	I	I
A	4		M	D	D	M	M
N	5		M	D	M	M	M
A	6		M	M	D	M	M

```
BANANA
PAN-DA
```

If only want distance, we don't need the full n^2 table space. Can just use 2 rows: the current row and the previous row. This will not give you information about the alignment of the strings.

Matrix Multiplication Order

February-28-13 2:46 PM

Matrix Multiplication Order

Given n matrices, want to compute the product
$M_1 M_2 M_3 \dots M_n$
Dimensions: $M_1: d_0 \times d_1$
$M_2: d_1 \times d_2$
$M_k: d_{k-1} \times d_k$

The number of multiplications required to multiply matrices of dimension $p \times q$ and $q \times r$ is pqr

Multiplying pairs of matrices in different orders will affect the number of multiplications required.

Solution

Store partial solution
$C[i, j] =$ min cost to compute $M_i \cdots M_j$

Base case:
$C[i, i] = 0$

$$C[i, j] = \min_{k \in \{i, \dots, j-1\}} \left(C[i, k] + C[k+1, j] + d_{i-1} d_k d_j \right)$$

Algorithm

```
for l = 1 to n - 1 do
    for i = 1 to n - l do
        j = i + l
```

$$C[i, j] = \min_{k \in \{i, \dots, j-1\}} \left(C[i, k] + C[k+1, j] + d_{i-1} d_k d_j \right)$$

Analysis

$O(n^3)$ time
$O(n^2)$ space

Memoization

February-28-13 3:37 PM

Perform dynamic programming, but instead of computing all previous results iteratively, perform the computation recursively from the end. At each recursive call, use the value in the cell table if it is available, otherwise compute and save the value.

Graph Search

February-28-13 3:49 PM

Graph Definition

A graph $G = (V, E)$
V: A set of vertices
$E \subseteq V \times V$: a set of edges
    $(u, v) \in E$ directed
    $uv \in E$ undirected

$n = |V|, m = |E|$

Representations

Adjacency Matrix

$n \times n$ matrix A

$$A[u, v] = \begin{cases} 1 & \text{if } (u, v) \in E \\ 0 & \text{otherwise} \end{cases}$$

$\Theta(n^2)$ space
Good for dense graphs - when m is close to n^2
Can check adjacency: $O(1)$ time
Enumerate all neighbours of u: $\Theta(n)$ time

Adjacency List

For each node, store a linked list of its neighbours.
Space:

$$O\left(n + \sum_{u \in V} |Adj(u)|\right) = O\left(n + \sum_{u \in V} \text{out-}\deg(u)\right) = O(n + m)$$

Enumerating all neighbours of u takes $\Theta(\text{out-}\deg(u))$ time.

Note

For a connected, directed graph

$$n - 1 \le m \le n(n-1)$$

For a connected, undirected graph

$$n - 1 \le m \le \frac{n(n-1)}{2}$$

Breadth First Search

Search nodes with FIFO ordering.
Get the following types of edges:

- Tree edges (edges to new nodes found by BFS)
- Forward edges (link to descendant node)
- Backward edges (link to ancestor node)
- Cross edges (link to already-found unrelated nodes)

Implementation

Given directed graph $G = (V, E)$ and $s \in V$, traverse all vertices readable from s

```
BFS(G, s) // use a queue Q
 1. for each v ∈ V mark v as undiscovered
 2. insert s to Q and mark s as discovered
 3. while Q is not empty
 4.       remove head u of Q
 5.       for each v ∈ Adj(u) do
 6.             if v is undiscovered then
 7.                   insert v to tail of Q
 8.                   mark v as discovered
 9.             Π[v] = u // Creates a tree rooted at s
```

Analysis

Lines 5-9 $O(\text{out-}\deg(u))$

Depth First Search

```
DFS(G, s)
mark s "discovered"
for each v ∈ Adj[s] do {
      if v is "undiscovered" then {
            DFS(G,v)
            Π[v] = s
      }
}
mark s as "finished"
```

Explore the whole graph

```
DFS(G)
for each v ∈ V, mark v "undiscovered"
for each v ∈ V do
      if v is "undiscovered" then DFS(G,v)
      Π[v] = s
```

Analysis

$$\Theta\left(n + \sum_{u \in V} \text{out} - \deg(u)\right) = \Theta(n + m)$$

Graph Problems

March-07-13 2:46 PM

Unweighted Shortest Path

Given directed graph $G = (V, E)$ and $s, t \in V$
Find path from s to t of shortest length (# of edges).

Bipartiteness / 2-colouring

Given undirected graph $G = (V, E)$ decide whether V can be partitioned into V_1, V_2 disjoint such that $\forall uv \in E\ u \in V_1$ and $v \in V_2$ or vice versa.

Topological Sort

Given a directed graph $G = (V, E)$ return a vertex order such that $\forall u, v \in E$, u appears before v.

Strongly Connected Components

Given a directed graph $G = (V, E)$
Partition V into components such that

u, v in some component $\Rightarrow \exists$ path $u \rightsquigarrow v$ and $v \rightsquigarrow u$

Unweighted Shortest Path Algorithm

1. Run BFS
2. Return path from s to t in BFS tree

Bipartition Algorithm

1. Run BFS(G,s)
2. For each $v \in V$
   a. colour v red if v is on odd level
   b. colour v blue if v is on blue level
3. For each $uv \in E$ if u, v same colour return no

$O(n + m)$ time

Topological Sort Algorithm

1. Run BFS
2. Record when each node is marked finished.
3. Output nodes in reverse order.

If a cycle is detected, fail.

$O(n + m)$

Correctness

Need to show $\forall (u, v) \in E$, u comes before v in reversed order.
$\forall (u, v) \in E$:
Case A:

u discovered first
BFS will go down edge $u \to v$ so v finished before u
u comes before v in reversed order

Case B:

v is discovered first. No cycle so $v ! \rightsquigarrow u$ so BFS below v finishes before searching u so v finished before u.
so u comes before v in reversed order.

SCC Algorithm

1. Run DFS(G)
2. Number vertices in order of finish
3. Form transposed graph G^T (reverse direction of each edge in G)
4. run DFS(G^T) preferring higher-numbered vertices
5. return DFS trees from 4

Correctness

Taka a DFS tree T of G^T
Let r be root, u be a node in T
Proof in 5 easy steps:

1. r has a higher number than u (if not, pick u first)
2. $\exists$ a path $u \to^* r$ in G (since $r \to^* u$ in G^T)
3. $\exists$ a path $r \to^* u$ in G:

   Suppose not.
   Case A:

   u discovered first in DFS(G)
   $\Rightarrow r$ finished before u
   $\Rightarrow$ Contradiction with 1

   Case B:

   r discovered first in DFS(G)
   $\Rightarrow$ again, r finished before u
   $\Rightarrow$ Contradiction with 1
4. $\forall u, v \in T\ \exists$ a path $u \leftrightsquigarrow v$
5. If $\exists$ a path $u \leftrightsquigarrow v \Rightarrow u, v$ in same tree
   If we find one of them in DFS of G^T will find the other one.

Min-Weight Spanning Tree

March-14-13 3:30 PM

Minimum Spanning Tree

Given a weighted undirected connected graph $G = (V, E)$, and weight function $w: E \to \mathbb{R}^+$. Want to find a connected subgraph T using all the vertices and minimizing the total weight of its edges.

Observation: The optimal subgraph must be acyclic.

Kruskal's Algorithm (Min-weight spanning tree)

```
T = ∅
repeat {
    pick next shortest edge e
    if T ∪ {e} does not contain a cycle
        insert e into T
}
```

More detailed implementation

1. Sort the edges w.r.t. weights (increasing)
2. Create set $\{v\}\ \forall v \in V$
3. for each edge uv in sorted order if u and v are in different sets then select uv and union the two sets.

Analysis

Line 1: $O(m \log m)$

Lines 4-5: Union-find data structure. Supports

1. Union of two disjoint sets
2. Find pointer to set containing given element
   $O(\alpha(n))$, amortized time (α is inverse Ackermann function)

Total time $O(m \log m) = O(m \log n)$

Graph is connected so $n - 1 \le m \le \frac{n(n-1)}{2}$

$$\log n \le \log m \le 2 \log n$$

Correctness

Lemma

The shortest edge e bweteen S and $V - S$ must be in the minimum spanning tree T^*

Proof

By contradiction. Suppose $e \notin T^*$. $T^* \cup \{e\}$ contains a cycle C

C contains another edge e' between S and $V - S$

Then $T^* \cup \{e\} - \{e'\}$ is a tree with weight $w(T^*) - w(e') + w(e) < w(T^*)$

Contradiction.

So each edge $e = uv$ inserted to T by Kruskal's algorithm is a correct MST edge.

Prim's Algorithm (1957)

```
S = {s},    T = ∅
while S ≠ V {
    pick shortest edge uv with u ∈ S, v ∈ V − S
    insert v into S, uv into T
```

Detailed Implementation

Maintain:

$$key[v] = \min_{u \in S} w(uv)\ \forall v \in V - S$$

$\pi[v] =$ the $u \in S$ obtaining this minimum

$Q = V - S$

```
 1. Q = V
 2. key[v] = ∞ ∀v∈ V-{S},  key[s]=0
 3. while Q ≠ ∅
 4.         pick v ∈ Q with smallest key[v]
 5.         print Π[v]v and remove v from Q
 6.         for each y ∈ Adj[v] do {
 7.                 if y ∈ Q and w(vy) < key[y] {
 8.                         key[y] = w(vw), Π[y] = v
 9.                 }
10.         }
11. }
```

Analysis

Option 1: No data structure

Line 4: $O(n)$, line 5 $O(1)$

Line 8: constant time, called $O(m)$ times

lines 6-8 take $O(\deg(v))$ time

Total:

$$O\left(n^2 + \sum_{v \in V} \deg(v)\right) = O(n^2 + m) = O(n^2)$$

Option 2: Store keys in a heap / priority queue

Supports EXTRACT-MIN in $O(\log n)$ time

CHANGE-KEY in $O(\log n)$

Total:

$$O\left(n \log n + \sum_{v \in V} \deg(v) \log n\right) = O(n \log n + m \log n) = O(m \log n)$$

Option 3: Use a Fibonacci Heap

- EXTRACT-MIN in $O(\log n)$
- DECREASE-KEY in $O(1)$ amortized time

Line 4: $O(\log n)$
Line 8: $O(1)$ amortized
Total:

$$O\left(n \log n + \sum_{v \in V} \deg v + O(1)\right) = O(n \log n + m)$$

Shortest Path

March-19-13 3:18 PM

Shortest Path

Given a weighted, directed graph $G = (V, E)$ and $s, t \in V$.
Weight $w: E \to \mathbb{R}^+$
Find path from s to t minimizing

$$w(p) = \sum_{e \in p} w(p)$$

All-pairs Shortest Path

Given weighted directed graph $G(V, E), v = \{1, \dots, n\}$.
Find shortest path between all pairs of vertices.
(assumption- no negative-weight cycles but negative-weight edges allowed)

Shortest path on DAG

Given weighted DAG $G = (V, E),\ s, t \in V$
Find shortest path from s to t

Dijkstra's Algorithm (1959)

```
// compute δ[v] = shortest path weight from s to v
S = {s}, δ[σ] = 0
while S != V do {
    pick edge (u,v) with u ∈ S and v ∈ V - S
        minimizing δ[u] + w(u, v)
    insert v into S
    δ[v] = δ[u] + w(u, v)
}
```

Implementation

Change Lines 7-8 from Prim's to

```
if y ∈ Q and key[v] + w(v,y) < key[y] {
    key[y] = key[v] + w(v,y), pi[y] = v
```

```
 1. Q = V
 2. key[v] = ∞ ∀v ∈ V − {s}, key[s] = 0
 3. while Q != ∅
 4.       pick v ∈ Q with smallest key
 5.       print π[v]v and remove v from Q
 6.       for each y ∈ Adj[v] do {
 7.             if y ∈ Q and key[v] + w(v,y) < key[y]
 8.                   key[y] = key[v] + w(v,y)
 9.       }
10. }
```

Runtime

$O(m + n \log n)$

Correctness

Claim:
If (u, v) is the edge from S to $V - s$ minimizing $\delta[u] + w(u, v)$
then $\delta[v] = \delta[u] + w(u, v)$

Proof:

i) There is a path from s to v of weight $\leq \delta[u] + w(u, v)$
ii) For any path p from s to v: $s \to^* u' \to v' \to^* v$
$$w(p) \geq \delta[u'] + w(u', v') \geq \delta[u] + w(u, v)$$

Thus $\delta[u] + w(u, v)$ ■

Each inserted vertex v has correct $\delta[v]$ (by Claim).
At the end, we have all the $\delta[v]'s$

All-Pairs Shortest Path

Method 0

Run Dijkstra starting from each vertex
$O\big(n(n \log n + m)\big)$ time
but requires all edges have positive weight.

Method 1: Dynamic Programming Solution

Define subproblems $(i = 1 \dots n, j = 1 \dots n, k = 0 \dots n - 1)$
$D[i, j, k] =$ min weight over all paths from i to j with at most k edges.
Answer: $D[i, j, n - 1]\ \forall i, j$

Base case:

$$D[i, j, 0] = \begin{cases} 0 & if\ i = j \\ \infty & otherwise \end{cases}$$

Recurrence:

$$D[i, j, k] = \min_{l \in \{1, \dots, k\}} \big(D[i, l, k - 1] + w(l, j)\big)$$

Analysis

$\Theta(n^3)$ entries, $\Theta(n)$ time each $\Rightarrow \Theta(n^4)$ time

Method 2: Same subproblems but only for k=1, 2, 4, 8, …

$$D[i, j, k] = \min_{l \in \{1, \dots, n\}} \left(D\left[i.l, \frac{k}{2}\right] + D\left[l, j, \frac{k}{2}\right]\right)$$

$\Theta(n^3 \log n)$ time

Method 3: Floyd-Warshall (1962)

Define subproblems
$D[i, j, k] =$ min weight over all paths from i to j with intermediate vertices in $\{1, \dots, k$
Base case:

$$D[i, j, 0] = \begin{cases} w(i, j) & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases}$$

Recurrence:
$D[i, j, k] = \min(D[i, j, k - 1], D[i, k, k - 1] + D[k, j, k - 1])$

Analysis $\Theta(n^3)$ entries, $\Theta(1)$ time per entry
Total: $\Theta(n^3)$ (smaller costs than Dijkstra)

DAG Shortest Path

Subproblems:
$\delta[v]$ = weight of shortest path from s to v
Answer: $\delta[t]$
Base case: $\delta[s] = 0$
$\delta[v] = \backslash$

Theory of NP-Completeness

March-26-13 2:46 PM

The Class P

P = all decision problems solvable in worst-case polynomial time (polytime)

Characteristics

- Decision problems: output should be "yes" or "no"
- Polynomial $O(n^d)$ for some d

The Class NP

NP = all decision problems that can be expressed in the form:

Input:

Object x

Output:

"yes" iff there exists object y such that property $R(x, y)$ holds where

1) object y has polynomial size
2) property $R(x, y)$ can be checked in polynomial time

y is called a **certificate**
R is called **verifier**

Hard Problems

3-colouring
0-1 knapsack
Largest simple path
Travelling Salesman Problem

For all of these problems, we don't know an algorithm that would run in $O(n^d)$ for any constant d.

Example Decision Problems

CYCLE

Input

Directed graph $G = (V, E)$

Output

"yes" iff there exists a cycle in G

TSP-DECIS

Input

Directed graph $G = (V, E)$ with weights $w: E \to \mathbb{R}_+$, and number W

Output

"yes" iff there exists a cycle that visits each vertex & has total weight $\le W$

0-1 KNAPSACK-DECIS

Input

$v_1, v_2, \dots, v_n, \quad w_1, w_2, \dots, w_n, \quad W, V$

Output

"yes" iff $\exists S \subseteq \{1, \dots, n\}$ s.t. $\sum_{i \in S} w_i \le W,\ \sum_{i \in S} v_i \ge V$

If 0-1 KNAPSACK-DECIS is hard then 0-1 KNAPSACK is also hard.

PRIME

Input

n-bit number N

Output

"yes" iff N is prime

Consider brute force algorithm:

```
for i = 2 … N
    if N is divisible by i return NO
return YES
```

⇒ RUNTIME is $O(Nn^2)$
N takes $n = \log_2 N + 1$ bits to write down
⇒ $O(2^n n^2)$ so not polynomial is the size of the input.

Not obvious but PRIME shown to be in P in 2002

Example NP Problems

TSP-DECIS

Certificate: Cycle C
Property to verify: C visits all vertices exactly once with total weight $\le W$
Checkable in polytime ⇒ TSP-DECIS is in NP

O-1 Knapsack

PRIME

- Not obvious if it is in NP (Pratt 1975 showed it is)
- In contrast, composite is easily shown to be in NP

NP-Complete

March-28-13 2:36 PM

Proposition

P ⊆ NP ⊆ EXPTIME

EXPTIME

Decision problems solvable in $O(2^{poly(n)})$

The Class NPC

Define "hardest" problems in NP
What does "L_1 easier than L_2" mean?
On way: "L_1 reduces to L_2"

Reduction

A **polytime reduction** from L_1 to L_2 is a **polytime** algorithm f such that the output of L_1 is "yes" on $x \Leftrightarrow$ the output of L_2 is "yes" on $f(x)$

We write $L_1 \le_p L_2$ iff there is a polytime reduction from L_1 to L_2.

To solve L_1

$x \to \boxed{\boxed{f} \xrightarrow{f(x)} \boxed{\text{solve for } L_2} \to} \to$ "yes" or "no"

Proposition A

If $L_1 \le_p L_2$ and $L_2 \le_p L_3$, then $L_1 \le_p L_3$

Proposition B

If $L_1 \le_p L_2$ and $L_2 \in P$ then $L_1 \in P$

NP-Complete (NPC)

L is NP-Complete iff
1) $L \in NP$
2) $\forall L' \in$ NP, $L' \le_p L$

Proposition C

Let L be an NP-complete problem
Then $L \neq P \Leftrightarrow P \neq NP$

SATISFIABILITY (SAT)

The first NPC problem
Input: Boolean formula on n variables
Output: "yes" iff there exists an assignment of Boolean values to $x_1, x_2, \dots, x_n$ such that $F(x_1, x_2, \dots, x_n)$ evaluates to true

Cook-Levin Theorem (1971)

SAT ∈ NP

Sketch Proof of Proposition

P ⊆ NP:
Have verification ignore certificate. $R(x, y) = R(x)$

NP ⊆ EXPTIME:
Try all possible certificates.
There are $2^{poly_1(n)}$ certificates. Takes $poly_2(n)$ time to evaluate each.
Total time: $O\left(2^{poly_1(n)}\, poly_2(n)\right)$

Example Reduction

Finding a median reduces to sorting.

Proof of Proposition C

Suppose $L \notin P$. Then $L \in NP - P$ by (1), so $P \neq NP$
Suppose $L \in P$. Then $\forall L' \in NP, L' \le_p L$
$\Rightarrow L' \in P$ by Prop. B So NP = P.

Example SAT Problem

$F(x_1, x_2, x_3) = \left(x_1 \leftrightarrow (\overline{x_2 \wedge \overline{x_3}})\right) \wedge \overline{x_2}$
YES. Set $x_1 = 1, x_2 = 0, x_3 = 0$ or 1

Sketch of Proof of Cook-Levin Theorem

1) SAT ∈ NP
   Certificate: The assignment (polysize)
   To verify: F evaluates to be true (polytime)
2) Need to give a reduction from every $L \in NP$
   Input: z
   Output: "yes" iff ∃y such that $R(z, y)$ is true where R can be checked by Algorithm d, which runs in polytime $p(n)$

   Idea: Simulate d by Boolean formula.
   Construct formula F as follows:
   Variables:
   $x[i, j]$ = the i^{th} bit in memory during the j^{th} step of the execution of d
   for $i = 1 \dots p(n),\ j = 1 \dots p(n)$
   Add clauses to relate $x[i, j]$ with $x[1, j-1] \dots x[p(n), j-1]$
   Add clauses for $x[i, 0]$, connect to values of y, z
   Take ∧ of all clauses.

More NPC Problems

April-02-13 2:39 PM

Recipe for proving NPC

Proposition D

If

1) L ∈ NP and
2) $L_0 \leq_P L$ for a known NP-complete problem L_0

then L is NP-complete.

3SAT

Input

Boolean formula F of the form

$$\bigwedge_{i=1}^{m} (\alpha_{i1} \vee \alpha_{i2} \vee \alpha_{i3})$$

where α_{ij} is wither a variable or its complement.

Output

"yes" iff there exists an assignment such that F evaluates to true.

This is NP-complete

Vertex Cover

Input

Undirected graph $G = (V, E)$, integer k

Output

"yes" iff ∃ subset $S \subseteq V$ of size k such that $\forall uv \in E, u \in S$ or $v \in S$

Independent Set (IS)

Input

Undirected graph $G = (V, E)$, integer k

Output

"yes" iff ∃ subset $S \subseteq V$ such that $\forall u, v \in S,\ uv \notin E$

Clique Problem (CLIQ)

Input

Undirected graph G, integer k

Output

"yes" iff ∃ subset $S \subseteq V$ such that $\forall u, v \in S\ uv \in E$

SUBSET-SUM Problem

Input

Set $a_1, \ldots, a_n$ of integers, W

Output

"yes" iff ∃ subset $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ such that

$$\sum_{l=1}^{k} a_{i_l} = W$$

Observation

Subset-sum $\leq_P$ 0/1 Knapsack-Decis

Proof

$a_1, \ldots, a_n, W \rightarrow$ weights $a_1, \ldots, a_n$ bound W
values $a_1, \ldots, a_n$ bound W

Proof of Proposition D

$\forall L' \in \text{NP},\ L' \leq_P L_0$ (since L_0 is NP-complete)
$L_0 \leq_P L$
$\Rightarrow L' \leq_P L$ by proposition A

Example 3SAT Formula

$F = (x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_3 \vee \overline{x_4})$
"yes" set $x_1 = x_2 = x_3 = 1,\ x_4 = 0$

3SAT is NP-Complete

3SAT is in NP since can verify using a certificate that is the variable assignment.

Now show SAT $\leq_P$ 3SAT
Given an arbitrary Boolean formula, write out tree where root nodes are variables (not negated) and internal nodes represent operations. Associate with each operation a new variable.
For example: $\phi \circ \psi$ where ϕ, ψ are Boolean formulae and x_ϕ, x_ψ, and $x_\circ$ represent ϕ, ψ, and $\circ$, respectively. Get expression $\left(x_\circ \leftrightarrow (x_\phi \circ x_\psi)\right)$
$\wedge$ these expressions together for each node, and also and with x_L, the variable representing the root node.

Now have conjunction of terms that may contain either
$\left(x \leftrightarrow (y \circ z)\right)$ or $(x \leftrightarrow \bar{y})$
Need to convert these into disjunction of literals where $\circ \in \{\wedge, \vee, \rightarrow, =\}$

Have to also add some dummy variables to the (x_L) term to give it three literals:

$$x_L \equiv (x_L \vee y \vee z) \wedge (x_L \vee y \vee \bar{z}) \wedge (x_L \vee \bar{y} \vee z) \wedge (x_L \vee \bar{y} \vee \bar{z})$$

This construction takes a polynomial amount of time. (Poly time to generate tree, constant time for each node in the tree).

Correctness

There exists an assignment making F' true ⟺ ∃ an assignment making F true

Vertex Cover $\leq_P$ Independent Set
Independent Set $\leq_P$ Vertex Cover

Reduction $(G, k) \rightarrow (G, n - k)$

Correctness: S independent set of G of size $\geq k$ ⟺ S^C vertex cover of size $\leq n - k$

IS $\leq_P$ CLIQ
CLIQ $\leq_P$ IS

Reduction $(G, k) \rightarrow (G^C, k)$

Correctness: S independent set in G ⟺ S is a clique in G^C

Independent Set is NP-Complete

Given a 3CNF formula with n variables, m clauses.
Construct G and k as follows:
for each clause $(\alpha_{i1} \vee \alpha_{i2} \vee \alpha_{i3})$ create 3 vertices v_{i1}, v_{i2}, v_{i3} and 3 edges $v_{i1}v_{i2},\ v_{i2}v_{i3},\ v_{i3}v_{i1}$
Whenever $\alpha_{ij} = \alpha_{i'j'}$ add "cross" edge $v_{ij}v_{i'j'}$

If the formula is satisfiable:
For each clause $\alpha_{i1} \vee \alpha_{i2} \vee \alpha_{i3}$ pick any j such that α_{ij} is true and put v_{ij} in S
Then $|S| = m = k$ and S is an independent set.

- check triangle edges
- check cross edges.

Given an independent set S of size $\geq k$ chose an assignment as follows:
whenever $v_{ij} \in S$, set α_{ij} to true
This assignment is consistent if $\alpha_{ij} = \overline{\alpha_{i'j'}}$ can't have both $v_{ij}, v_{i'j'} \in S$
For each triangle, at most one v_{ij} in S but since $|S| \geq m$, exactly one v_{ij} is in S so $\alpha_{i1} \vee \alpha_{i2} \vee \alpha_{i3}$ is true for each. ∎

Consequence:
IS, VS, CLIQ are NP-complete.

Reduction from VC to Subset-Sum

Build an incidence matrix with an extra column of all 1's

Example:

		e_4	e_3	e_2	e_1	e_0	=	
v_1	1	0	0	0	1	1		a_1
v_2	1	1	0	1	1	0		a_2
v_3	1	0	1	1	0	1		a_3
v_4	1	0	1	0	0	0		a_4
v_5	1	1	0	0	0	0		a_5
						1		b_0

					1	0		b_1
⋮								
		1	0	0	0	0		b_4

Row is a value in the set.
Want to get sum $W = K22222$

More formally
Given $G = (V, E)$, integer K
Let $c_{ij} = \begin{cases} 1 & \text{if } e_j \text{ incident to } v_j \\ 0 & \text{otherwise} \end{cases}$
Construct numbers $a_1, \ldots, a_k, b_0, \ldots, b_{m-1}, W$

$$a_i = 10^m + \sum_{j=0}^{m-1} c_{ij} \times 10^i$$

$$b_j = 10^j$$

$$W = K \cdot 10^m + \sum_{j=0}^{m-1} 2 \cdot 10^j$$

Correctness

$\exists$ vertex cover S of G of size $K \Leftrightarrow \exists$ subset T of $\{a_1, \ldots, a_n, b_0, \ldots, b_{m-1}\}$ summing to W

Proof

$\Rightarrow$ given S
choose $T = \{a_i : v_i \in S\} \cup \{b_j : e_j \text{ is incident to exactly on vertex of } S\}$
Then sum of T is W by construction of S

$\Leftarrow$ Given T
Choose $S = \{v_i : a_i \in T\}$
Then $|S| = k$ because of the m-th digit and
e_i is incident to one or two vertices m because of the j-th digit. ■

Consequence

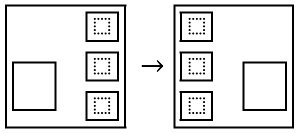
Subset-sum, Knapsack are NP-complete

Beyond NP

April-04-13 3:00 PM

PSPACE-Complete Example

Warehouseman's problem



Unsolvable Example

Halting Problem

Input

string P representing a problem, string x

Output

"yes" iff P halts on x

Theorem (Turing 1936)

Halting is undecidable

Proof

Suppose you claim to have an algorithm $f(P, x)$ for Halting
I claim your algorithm is wrong on input (P, P)

```
Let P =
"main (string Q) {
 1. if f(Q, Q) = yes then
     a. while (1) do nothing
 2. else return
}"
```

Case 1: $f(P, P) =$ yes
Then P would halt on input P by line 2. Contradiction
Case 2: $f(P, P) =$ no
Then P hals on input P by line 3. Contradiction