Dynamic Programming

- Characterize structure of optimal solution
- Recursively define value of optimal solution
- Compute value of optimal solution bottom-up
- Construct an optimal solution from this

Problems that need such solution:

- Matrix chain multiplication
- Optimal polygon triangulation
- Longest common subsequence

Key ingredients to look for:

- Optimal substructures
- Overlapping subproblems

Matrix multiplication problem

- Multiply A1 x A2 x A3
- Dimensions: (pXq) x (qXr) x (rXs)
- Count of Operations:
  - (A1 X A2) X A3 -> option-1 = pqr + prs
  - A1 X (A2 X A3) -> option-2 = qrs + pqs

The order of the matrices in a matrix chain is such that the multiplication is compatible.

Polygon triangulation Problem

- **Cost of triangulation** is considered to be the sum of side lengths of the triangles
- **Substructures** - The polygon can be split into two subpolygons by joining two vertices
- **Overlapping** - A pentagon can be split into triangle and quadrilateral and the resulting quadrilateral splits further into two triangles, which two can as well be the starting point

Matrix Chain Multiplication Problem

- Let no of alternative parenthesizations be denoted by P(n)
- A sequence of n matrices can be split between k-th and (k+1)-th matrices for any k=1,2,..,n-1 and then parenthesize the two resulting subsequences independently.
- $P(n) = \sum P(k)\, P(n-k)$ if $n \geq 2$ with $P(1)=1$
- Solution is $C(n) = (1/n+1)\ {}^{2n}C_n = (4^n/n^{3/2})$,
- This is called Catalan number and is exponential.
- Hence no of solutions is exponential and exhaustive search is a poor strategy

Catalan number

- Balanced parentheses
- Mountain ranges
- Diagonal avoiding paths
- Polygon triangulation
- Hands across a table
- Rooted Binary trees
- Planar trees
- Skewed polyominos
- Multiplication ordering

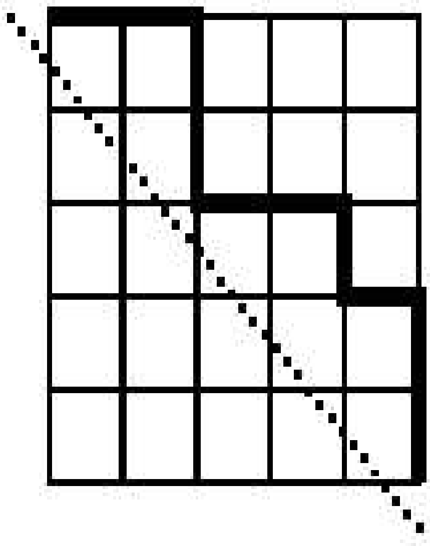
Balanced parentheses

$n = 0$:	*	1 way
$n = 1$:	()	1 way
$n = 2$:	()(), (())	2 ways
$n = 3$:	()()(), ()(()), (())(), (()()), ((()))	5 ways
$n = 4$:	()()()(), ()()(()), ()(())(), ()(()()), ()((())), (())()(), (())(()), (()())(), ((()))(), (()()()), (()(())), ((())()), ((()())), (((())))	14 ways
$n = 5$:	()()()()(), ()()()(()), ()()(())(), ()()(()()), ()()((())), ()(())()(), ()(())(()), ()(()())(), ()((()))(), ()(()()()), ()(()(())), ()((())()), ()((()())), ()(((()))), (())()()(), (())()(()), (())(())(), (())(()()), (())((())), (()())()(), (()())(()), ((()))()(), ((()))(()), (()()())(), (()(()))(), ((())())(), ((()()))(), (((())))(), (()()()()), (()()(())), (()(())()), (()(()())), (()((()))), ((())()()), ((())(())), ((()())()), (((()))()), ((()()())), ((()(()))), (((())())), (((()()))), ((((()))))	42 ways

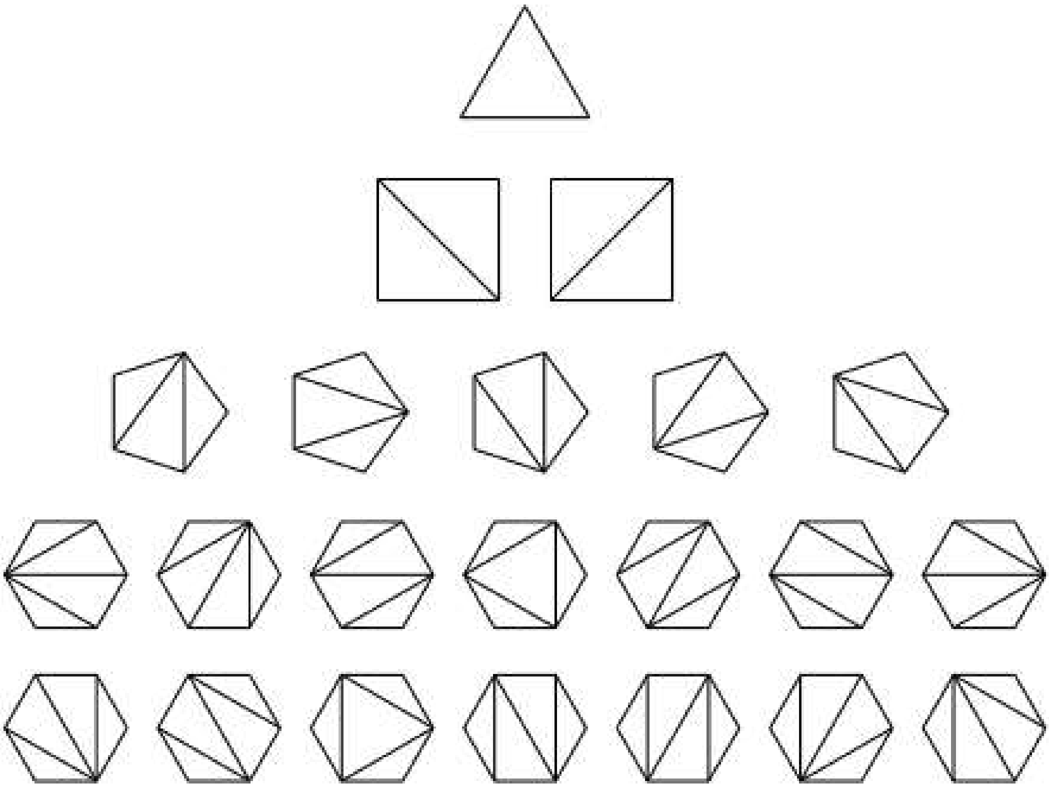
Mountain ranges

<table>
<tr><td>$n = 0$:</td><td><pre>*</pre></td><td>1 way</td></tr>
<tr><td>$n = 1$:</td><td><pre>/\</pre></td><td>1 way</td></tr>
<tr><td>$n = 2$:</td><td><pre>   /\
/\/\, / \</pre></td><td>2 ways</td></tr>
<tr><td>$n = 3$:</td><td><pre>   /\
           /\   /\   /\/\   / \
/\/\/\, /\/ \, / \/\, /   \, /   \</pre></td><td>5 ways</td></tr>
</table>

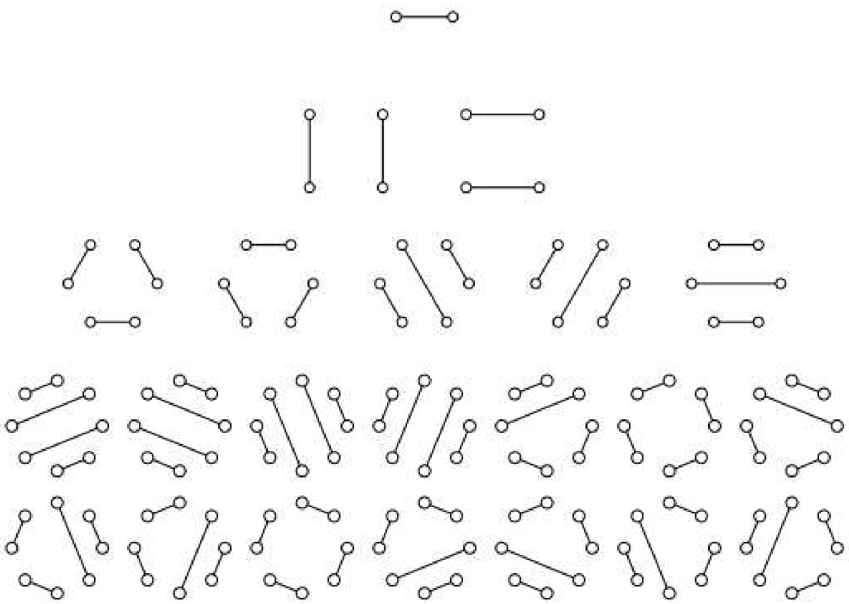
Diagonal avoiding paths



Polygon triangulation

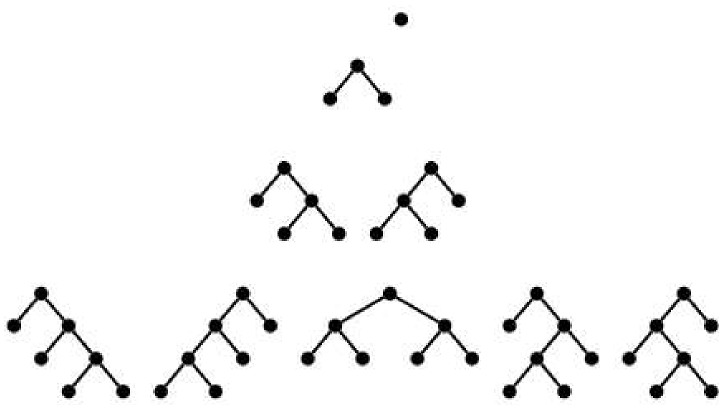


Hands across a table

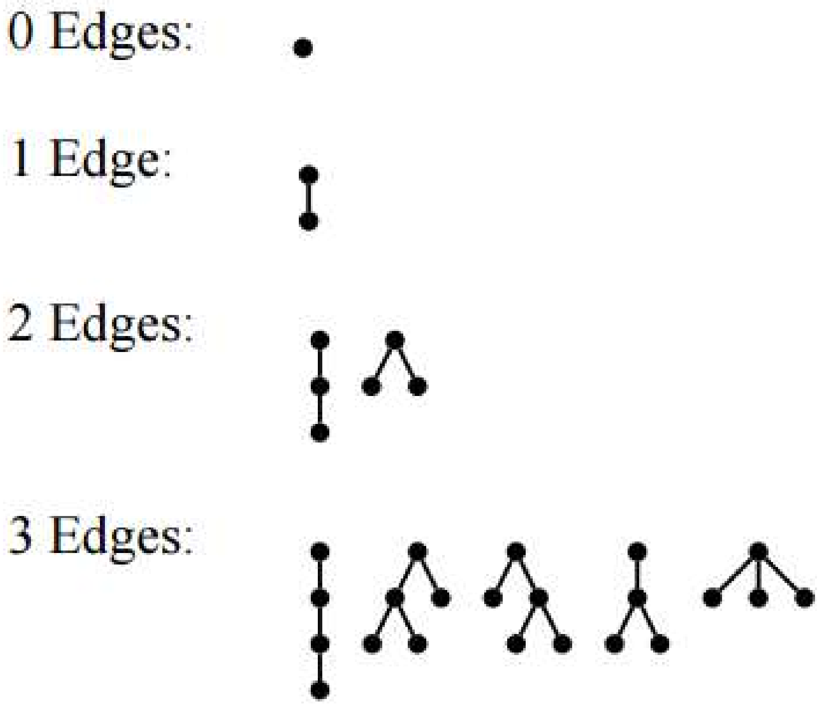


Rooted binary trees

(internal nodes – those which connect to two nodes below)

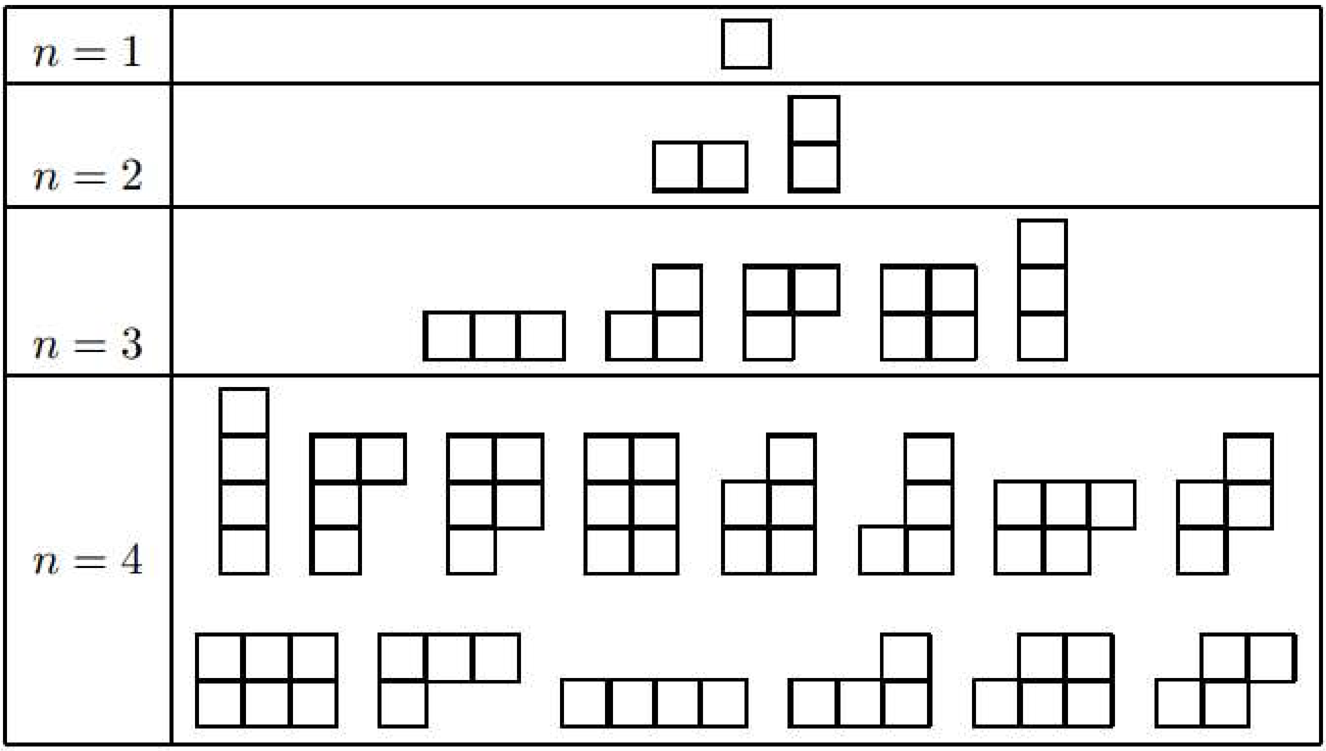


Plane rooted trees



Skew polyominos

(with perimeter 2n+2)



Matrix chain multiplication

$n=0$	(a)	1 way
$n=1$	$(a \cdot b)$	1 way
$n=2$	$((a \cdot b) \cdot c)$, $(a \cdot (b \cdot c))$	2 ways
$n=3$	$(((a \cdot b) \cdot c) \cdot d)$, $((a \cdot b) \cdot (c \cdot d))$, $((a \cdot (b \cdot c)) \cdot d)$, $(a \cdot ((b \cdot c) \cdot d))$, $(a \cdot (b \cdot (c \cdot d)))$	5 ways
$n=4$	$((((a \cdot b) \cdot c) \cdot d) \cdot e)$, $(((a \cdot b) \cdot c) \cdot (d \cdot e))$, $(((a \cdot b) \cdot (c \cdot d)) \cdot e)$, $((a \cdot b) \cdot ((c \cdot d) \cdot e))$, $((a \cdot b) \cdot (c \cdot (d \cdot e)))$, $(((a \cdot (b \cdot c)) \cdot d) \cdot e)$, $((a \cdot (b \cdot c)) \cdot (d \cdot e))$, $((a \cdot ((b \cdot c) \cdot d)) \cdot e)$, $((a \cdot (b \cdot (c \cdot d))) \cdot e)$, $(a \cdot (((b \cdot c) \cdot d) \cdot e))$, $(a \cdot ((b \cdot c) \cdot (d \cdot e)))$, $(a \cdot ((b \cdot (c \cdot d)) \cdot e))$, $(a \cdot (b \cdot ((c \cdot d) \cdot e)))$, $(a \cdot (b \cdot (c \cdot (d \cdot e))))$	14 ways

Generating function - Fibonacci

- Consider the generating function to be power series with Fibonacci numbers as coefficients
- $F(z) = \Sigma F_i z^i$ with $F_i = F_{i-1} + F_{i-2}$
- $F(z) = \Sigma (F_{i-1} + F_{i-2}) z^i$
- $= z + z \Sigma F_{i-1} z^{i-1} + z^2 \Sigma F_{i-2} z^{i-2}$
- $= z + z F(z) + z^2 F(z)$
- $= z / (1 – z - z^2)$

Generating function – Catalan number

- $C(n) = \Sigma\ C(k)C(n-k)$ summed over all k=1 to n-1
- $C(n)=C_{n-1}C_0+C_{n-2}C_1+ \ldots + C_1C_{n-2}+C_0C_{n-1}$
- Using generating function, $f(z) = \Sigma\ C(n)\ z^n$
- Next, $[f(z)]^2= C_0C_0+(C_1C_0 + C_0C_1)z+ \ldots= C_1 + C_2z+C_3 z^2+..$
- Multiplying by z on both sides, $f(z)= C_0+z[f(z)]^2$
- This quadratic in f(z) solves to $(1 - \surd(1-4z))/2z$

Solution for Catalan number

$$(1-4z)^{1/2} = 1 - \frac{\left(\frac{1}{2}\right)}{1}4z + \frac{\left(\frac{1}{2}\right)\left(-\frac{1}{2}\right)}{2\cdot 1}(4z)^2 - \frac{\left(\frac{1}{2}\right)\left(-\frac{1}{2}\right)\left(-\frac{3}{2}\right)}{3\cdot 2\cdot 1}(4z)^3 + \frac{\left(\frac{1}{2}\right)\left(-\frac{1}{2}\right)\left(-\frac{3}{2}\right)\left(-\frac{5}{2}\right)}{4\cdot 3\cdot 2\cdot 1}(4z)^4 - \frac{\left(\frac{1}{2}\right)\left(-\frac{1}{2}\right)\left(-\frac{3}{2}\right)\left(-\frac{5}{2}\right)\left(-\frac{7}{2}\right)}{5\cdot 4\cdot 3\cdot 2\cdot 1}(4z)^5 + \cdots$$

We can get rid of many powers of 2 and combine things to obtain:

$$(1-4z)^{1/2} = 1 - \frac{1}{1!}2z - \frac{1}{2!}4z^2 - \frac{3\cdot 1}{3!}8z^3 - \frac{5\cdot 3\cdot 1}{4!}16z^4 - \frac{7\cdot 5\cdot 3\cdot 1}{5!}32z^5 - \cdots$$

$$f(z) = 1 + \frac{1}{2!}2z + \frac{3\cdot 1}{3!}4z^2 + \frac{5\cdot 3\cdot 1}{4!}8z^3 + \frac{7\cdot 5\cdot 3\cdot 1}{5!}16z^4 + \cdots$$

Solution for Catalan number

The terms that look like $7 \cdot 5 \cdot 3 \cdot 1$ are a bit troublesome. They are like factorials, except they are missing the even numbers. But notice that $2^2 \cdot 2! = 4 \cdot 2$, that $2^3 \cdot 3! = 6 \cdot 4 \cdot 2$, that $2^4 \cdot 4! = 8 \cdot 6 \cdot 4 \cdot 2$, et cetera. Thus $(7 \cdot 5 \cdot 3 \cdot 1) \cdot 2^4 4! = 8!$. If we apply this idea to Equation we can obtain:

$$f(z) = 1 + \frac{1}{2}\left(\frac{2!}{1!1!}\right)z + \frac{1}{3}\left(\frac{4!}{2!2!}\right)z^2 + \frac{1}{4}\left(\frac{6!}{3!3!}\right)z^3 + \frac{1}{5}\left(\frac{8!}{4!4!}\right)z^4 + \cdots = \sum_{i=0}^{\infty} \frac{1}{i+1}\binom{2i}{i} z^i.$$

From this we can conclude that the i^{th} Catalan number is given by the formula

$$C_i = \frac{1}{i+1}\binom{2i}{i}.$$

Solution for Catalan number –use Stirling approximation on factorials

- Numerator= $(2n/e)^{2n} \sqrt{2\prod 2n}$
- Denominator =$(n) [(n/e)^n \sqrt{2\prod n}]^2$
- Overall expression will become
  - $(1/\sqrt{\prod})4^n / n^{(3/2)}$
    - **This implies that Catalan number grows exponentially**

Matrix Chain Order

```
MATRIX-CHAIN-ORDER (p)
    n=length[p]-1
    for i=1 to n
        m[i,i]=0
    for l=2 to n
        for i=1 to n-l+1
            j=i+l-1
            m[i,j]=INF
            for k=i to j-1
                q=m[i,k]+m[k+1,j]+p_{i-1}p_k p_j
                if q< m[i,j]
                    m[i,j]=q
                    s[i,j]=k
Return m and s
```

Multiplication of the chain

```
Matrix-Chain-Multiply (A,s,i,j)
   if j>i
        X=M-C-M(A,s,i,s[i,j])
        Y=M-C-M(A,s,s[i,j]+1,j)
        return MATRIX-MULTIPLY(X,Y)
   else
        return A_i
```

Memoized Matrix Chain

```
MMC(p)
    n=length[p]-1
    for i= 1 to n
        for j= i to n
            m[i,j]=INF
    return LOOK-UP-CHAIN(p,1,n)

LookUpChain(p,i,j)
    if m[i,j]<INF return m[i,j]
    if i==j
        m[i,j]=0
    else
        for k=i to j-1
            q=LUC(p,i,k)+ LUC(p,k+1,j)+ p_{i-1}p_k p_j
            if q < m[I,j]
                m[I,j]=q
Return m[i,j]
```

Example of Matrix Chain

$m[i,j]=\min\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\}$ for all $i \le k < j$

Suppose, the best parentheses of 2-5 is needed.

$m[2,5]= m[2,2]+m[3,5]+p_1p_2p_5$

$m[2,5]= m[2,3]+m[4,5]+p_1p_3p_5$

$m[2,5]= m[2,4]+m[5,5]+p_1p_4p_5$

Here, all smaller chain matrix results are stored.

The minimum among the three is stored in m[][]

The corresponding index k is stored in s[][]

Recursive Matrix Chain

```
RMC(p,i,j)
    if i==j then return zero
    m[i,j]=INF
    for k=i to j-1
        q=RMC(p,i,k)+RMC(p,k+1,j)+p_{i-1}p_k p_j
        if q<m[i,j]
            m[i,j]=q
Return m[i,j]
```

$T(n) \geq 1+\sum(T(k)+T(n-k)+1) \geq 2 \sum T(i) + n \geq 2^{n-1}$

Greedy Algorithm for Activity Selection

- Locally optimum choice leading to globally optimum solution
- Greedy choice property & Optimal substructure
- If A is optimal then A’=A-{1} is also optimal solution for the set S’ with $s_i \geq f_1$
- Induction at every step - top down approach iteratively solves a smaller subproblem

Greedy Activity Selector

- To accommodate maximum number of activities with set of start times and finish times
- Sort the activities first on finish times in order to maximize the amount of unscheduled time remaining

```
    n=length[S]
    A={1}
    j=1
    for i=2 to n
        if s_i ≥ f_j   //Compatibility check
                A=A U {i}
                j=i  // most recent addition
return A
```

Knapsack problem

- Thief robs a store having n items with value v_i and weight w_i with total carrying capacity of W.
- Optimal substructure – if a portion or whole of the most valuable is taken out, the remaining load is to be selected from remaining items
- In Fractional knapsack, use top-down greedy strategy on unit price of items $u_i = v_i / w_i$
- In 0/1 knapsack, the strategy fails as there can be empty space left out – use dynamic programming to solve the resulting overlapping subproblems bottom-up

Data compression - Huffman coding

- Variable length encoding based on frequency of occurrence of the symbols
- Collapse two least occurring symbols into compound symbol
- Continue the process until two symbols are left
- Heap based construction yields the coding tree
- Minimum overhead on average code word length ensured by collapsing of least probable symbols
- No other code that uses any other strategy is capable of better compression

Greedy Algorithm on Matroids

- Matroid is an ordered pair M= [S,I]
- To find Maximal weight independent subset I of a set S having elements of weight w

Graphic Matroid theory

Generic Matroid

S is a finite non-empty set

Independent: I is non-empty family of subsets of S

Hereditary: If B ε I and A is contained in B then A ε I

Exchange: If A,B ε I and |A| < |B| then some x ε B-A exists such that A U {x} ε I

Graphic Matroid

Set of edges E of a graph (V, E)

A is I iff A is acyclic, Set of edges are independent iff it is forest

Depletion of an edge retains the independent property – subset of forest is a forest

Extension to AU{x} possible till formation of cycle

Maximal Independent Subset

- When no more extensions are possible
- All maximal subsets are of same size
- Add the edge weights to get w(A) = ∑ w(x)
- Find maximum weight independent subset A of weighted matroid

```
GREEDY(M,w)
    A = NULL
    sort S[M] into non-decreasing order by weight
    for each x ε S[M] taken in order of w
          if A U {x} ε I[M]
                    A = A U {x}
Return A
```

Exchange property of graphic matroid

- Suppose A and B are forests of G and that |B| > |A|, i.e. A, B are acyclic sets of edges and B contains more edges.
- Now, forest with k edges contains exactly |V| - k trees. Begin with |V| trees and no edges. As and when an edge is introduced, no of trees reduce by 1. So forest A contains |V|-|A| trees and forest B has |V|-|B| trees.
- Since forest B has fewer trees, it must be having some tree T whose vertices are in two different trees in forest A.

Exchange property of graphic matroid

- Now, T being connected, it must have an edge (u,v) connecting vertices in two different trees in forest A. Thus edge (u,v) can be added to A without creating a cycle. This satisfies exchange property.

- An edge e is an extension of A iff e is not in A and addition of e does not create a cycle. When no more extension is possible, A is maximal. For this A should not be contained in any larger independent subset of M.

Size of independent subsets

- **All maximal independent subsets in a matroid have the same size.**

- Suppose on the contrary, B is a maximal independent subset that is larger than another one A. In that case, exchange property implies that A is extendable to A U {x} for some x ε B-A. which contradicts the assumption that A is maximal.

Correctness of greedy algorithm

- **Lemma 1:** Let x be the first element of S (sorted on weight) such that {x} is independent. If such x exists, then there exists an optimal subset A of S that contains x. (greedy choice property)
- **Lemma 2:** If x is not an extension of NULL, it is not an extension of any independent subset A of S.
- **Lemma 3:** Let x be first element chosen. The remaining problem becomes M'=(S',I') where S'={y ε S: {x,y} ε I} and I'={B is subset of S-{x}: B U {x} ε I} i.e. M' is contraction of M by x.

Proof for Lemma 1

- If no x exists, we have empty set – independent.
- Otherwise, B is any non-empty optimal subset. Assume that x does not belong to B. otherwise A=B and we are done. No element of B has weight > w(x). Observe that y ε B implies that {y} is independent. Since B ε I and I is hereditary.
- Our choice of x therefore ensures w(x) ≥ w(y) for any y ε B. Now construct set A as follows. Begin with A={x} which by choice of x makes A independent.
- Using exchange property repeatedly, find a new element of B that can be added to A until |A|=|B| while preserving independence of A.
- Then A= B-{y}U{x} for some y ε B and so we have w(A)=w(B) – w(y) + w(x) ≥ w(B). Because B is optimal, A must also be optimal and since x εA, lemma is proven.

Proof for Lemma 2 and 3

- **Proof:** Assume that x is an extension of A but not of NULL. Since x is extension of A, A U {x} ε I. Since I is hereditary, {x} ε I which contradicts the assumption.
- **Proof:** If A is any maxwt independent subset of M containing x, then A'=A-{x} is an independent subset of M'. Conversely, any independent subset A' of M' yields an independent subset A=A'U{x} of M. Since in both cases we have w(A) = w(A') + w(x), a maxwt solution in M containing x yields maxwt solution in M' and vice versa.

Proof for matroid greedy theorem

- By Lemma 2, any element that is not an initial extension of NULL, can be forgotten.
- Once first element x is selected, Lemma 1 implies that the greedy algorithm does not err by adding x to A, since there exists an optimal subset containing x.
- Finally Lemma 3 implies that the problem gets reduced to one of finding an optimal subset in M’ that is a contraction of M by x.
- After setting A = {x}, the remaining steps act on M’=(S’,I’) because B is independent in M’ iff B U {x} is independent in M, for all sets B ε I’.
- Thus subsequent operations of greedy algorithm finds maxwt independent subset for M’ and overall operation finds a maxwt independent subset for M.

Task scheduling problem

- S={1,2,…,n} of n unit-time tasks with deadlines $\{d_1,d_2,\ldots,d_n\}$ and penalty (non –ve) $\{w_1,w_2,\ldots,w_n\}$ for missing the deadlines. To find a schedule for S that minimizes the total penalty.
- Early-deadline-first schedule is possible by swapping tasks while constructing a schedule starting from NULL set. Then we are left with a subset A of early tasks where tasks meet deadlines sorted in order of non-decreasing deadlines and another subset {S-A} of late tasks where tasks miss deadlines and these may appear in any order.

Early and late task sets

- Set A of tasks is independent if there exists a schedule where none is late, the set of early tasks for a schedule forms an independent set of tasks.
- Let I denote the set of all independent sets. For t=1,2,..,n; let $N_t(A)$ denote no of tasks in A whose $d_i \le t$ i.e. deadline is t or earlier.
- Clearly, if $N_t(A) > t$ for some t, then there is no way to make a schedule with no late tasks for set A, because there are more than t tasks to finish before time t.
- Hence set A is independent implies that $N_t(A) \le t$ for t=1,2,..,n.

Minimizing penalty on early tasks

- Then there is no late task if those in A are scheduled in order of non-decreasing d_i. The i-th largest deadline is at most i. Given these properties it is easy to compute whether a given set of tasks is independent.
- Minimizing sum of penalties on late tasks is equivalent to maximizing the penalty on early tasks. So we can use greedy strategy to find an independent set A of tasks that maximizes the total penalty. Then this system must be shown to be a matroid.

Independent task sets

- Every subset of an independent set of tasks is certainly independent.
- Suppose B, A are independent with |B| > |A|.
- Let k be the largest t so that $N_t(B) \le N_t(A)$.
- Since $N_n(B)=|B|$ and $N_n(A)=|A|$, but |B| > |A|,
- we must have $k < n$ and $N_j(B) > N_j(A)$ for all j in the range $k+1 \le j \le n$.
- Therefore B contains more tasks with deadline k+1 than A does.

Exchange property of tasks

- Let x be a task in B-A with deadline k+1. Let A'=A U {x}.
- To show exchange property, we need to show that A' must be independent, using above property.
- For the range $1 \le t \le k$, $N_t(A') = N_t(A) \le t$ since A is independent.
- For the range $k < t \le n$ we have $N_t(A') \le N_t(B) \le t$ since B is independent. Thus A' is independent.

Task scheduling - example

Task	1	2	3	4	5	6	7
Deadline	4	2	4	3	1	4	6
Penalty	70	60	50	40	30	20	10

Augment A	**Deadline**	**$N_1(A)$ ≤ 1**	**$N_2(A)$ ≤ 2**	**$N_3(A)$ ≤ 3**	**$N_4(A)$ ≤ 4**	**$N_5(A)$ ≤ 5**	**$N_6(A)$ ≤ 6**	**Remarks**
{1}	4	0	0	0	1	-	-	Independent
{1,2}	4,2	0	1	1	2	-	-	-do-
{1,2,3}	4,2,4	0	1	1	3	-	-	-do-
{1,2,3,4}	4,2,4,3	0	1	2	4	-	-	-do-
{1,2,3,4,5}	4,2,4,3,1	1	2	3	5	-	-	**$N_4(A) > 4$**
{1,2,3,4,6}	4,2,4,3,4	0	1	2	5	-	-	**$N_4(A) > 4$**
{1,2,3,4,7}	4,2,4,3,6	0	1	2	4	4	5	Independent

Result of scheduling example

- The set S={1,2,3,4,5,6,7} of tasks is sorted on the penalty.
- Next, we sort A on deadlines so that schedule is early tasks <2,4,1,3,7> followed by late tasks <5,6>
- Final schedule <2,4,1,3,7,5,6> with penalty=50