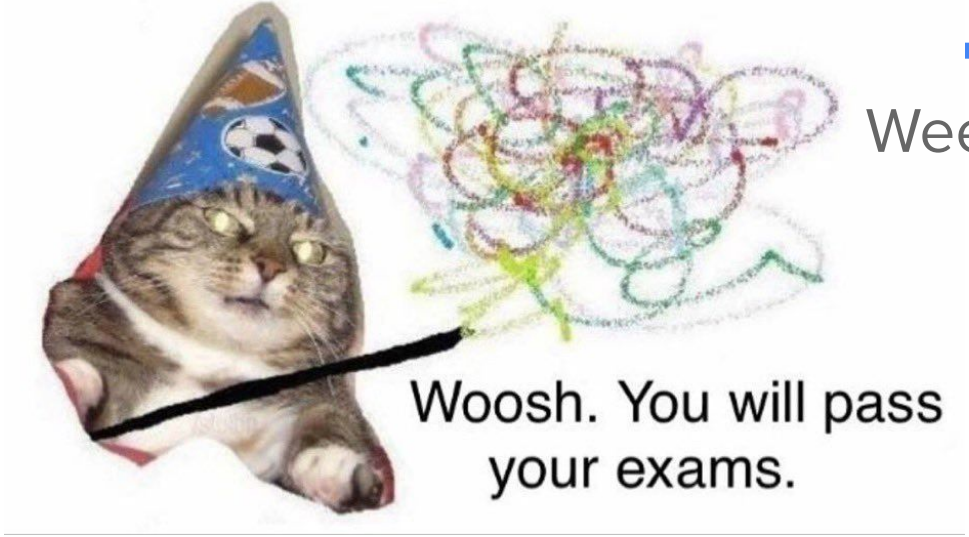
CS180 Discussion



Week 10

Recap

Coins

Consider a row of n coins of values v1,⋯ ,vn, where n is even. We play a game against an opponent by alternating turns (you can both see all coins at all times) . In each turn, a player selects either the first or last coin from the row, removes it from the row permanently, and receives the value of the coin. Determine the maximum possible amount of money we can win if we move first.

Example 1: [5, 3, 7, 10] : The user collects maximum value as 15(10 + 5) - Sometimes the greedy strategy works

Example 2. [8, 15, 3, 7] : The user collects maximum value as 22(7 + 15) – In general the greedy strategy does not work

Coins

[8, 15, 3, 7, 6, 3, 9, 4]

Coins

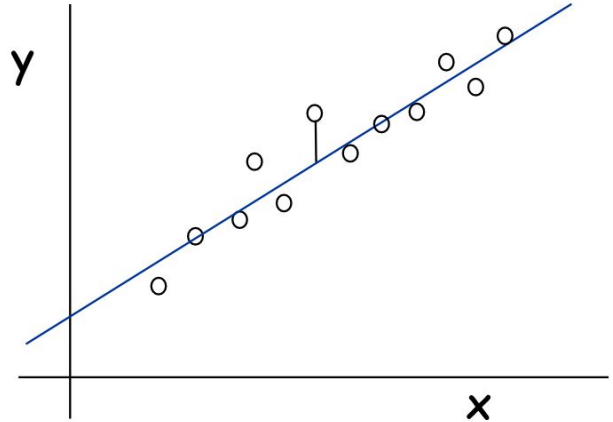
[8, 15, 3, 7, 6, 3, 9, 4]

$$F(i,j) = \text{Max}(Vi + \min(F(i+2,j), F(i+1,j-1)),\ Vj + \min(F(i+1,j-1), F(i,j-2)))$$

Least Squares

Least squares is the problem of fitting a line to a set of points, while minimizing the sum of squared error.

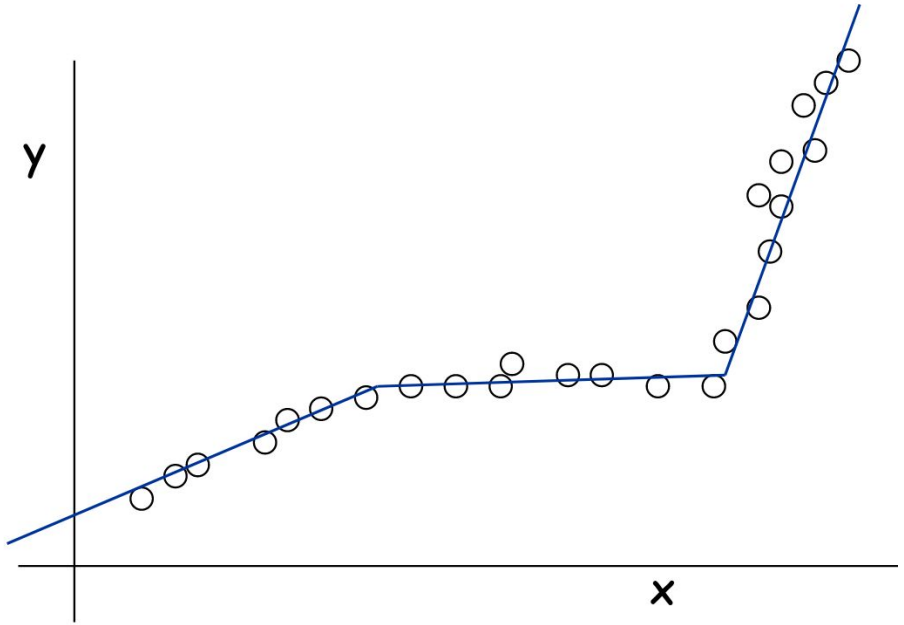
$$SSE = \sum_{i=1}^{n} (y_i - ax_i - b)^2$$



Segmented Least Squares

Segmented least squares is the problem of fitting a set of lines to a set of points, while minimizing the the tradeoff function:

SSE + cL



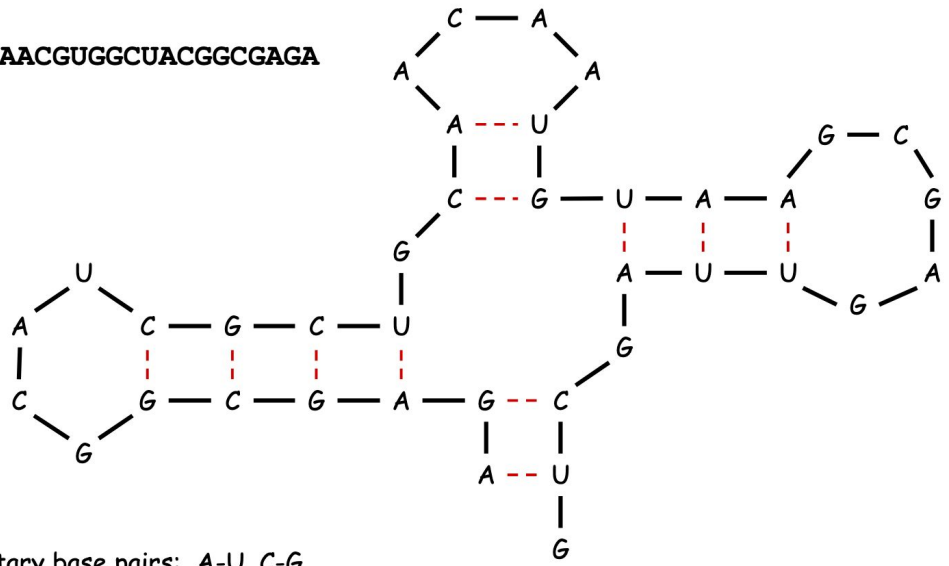
Segmented Least Squares

```
INPUT: n, p_1,…,p_N , c

Segmented-Least-Squares() {
   M[0] = 0
   for j = 1 to n
      for i = 1 to j
         compute the least square error e_ij for
         the segment p_i,…, p_j

   for j = 1 to n
      M[j] = min_{1 ≤ i ≤ j} (e_ij + c + M[i-1])

   return M[n]
}
```

Max RNA Base Folding

- each pair are separated by at least 4 intervening bases
- If (bi, bj) and (bk, bl) are two pairs in S, then we cannot have i < k < j < l

Ex: GUCGAUUGAGCGAAUGUAACAACGUGGCUACGGCGAGA



complementary base pairs: A-U, C-G

Max RNA Base Folding

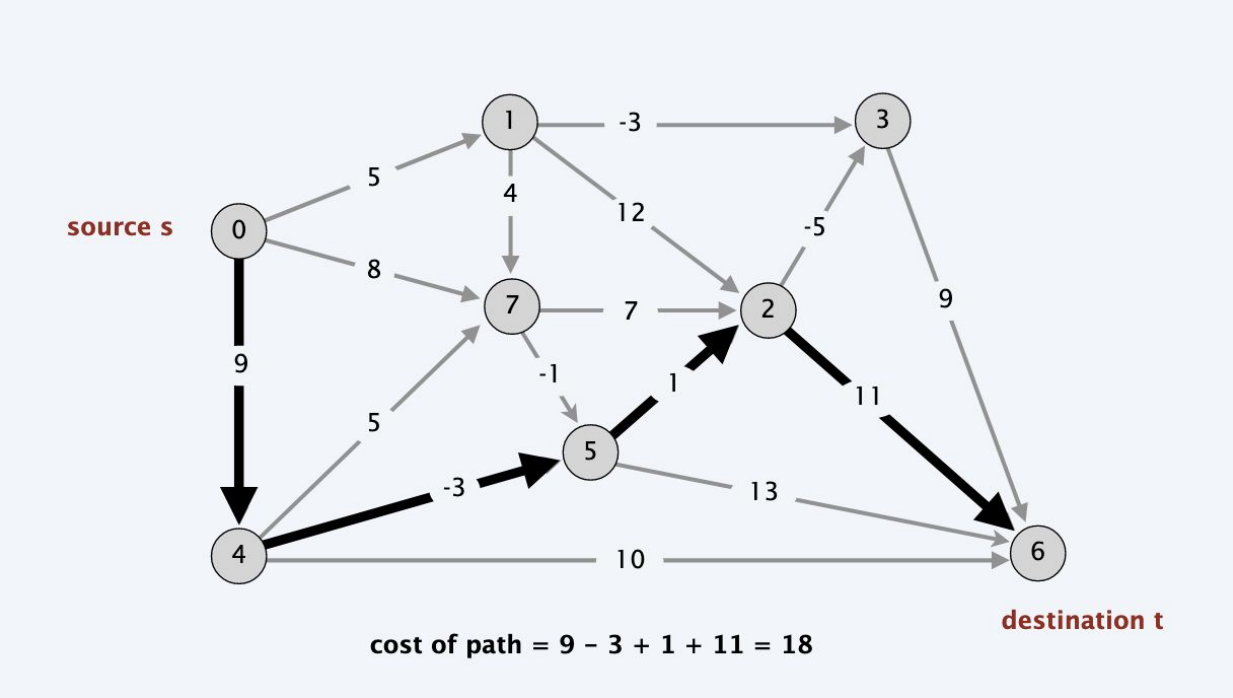
Notation. OPT(i, j) = maximum number of base pairs in a secondary structure of the substring $b_i b_{i+1} \ldots b_j$.

- Case 1. If $i \geq j - 4$.
  - OPT(i, j) = 0 by no-sharp turns condition.
- Case 2. Base b_j is not involved in a pair.
  - OPT(i, j) = OPT(i, j-1)
- Case 3. Base b_j pairs with b_t for some $i \leq t < j - 4$.
  - non-crossing constraint decouples resulting sub-problems
  - $OPT(i, j) = 1 + \max_t \{ OPT(i, t-1) + OPT(t+1, j-1) \}$

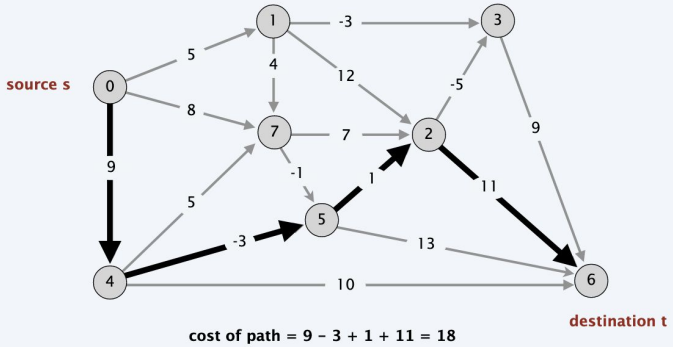
take max over t such that $i \leq t < j-4$ and b_t and b_j are Watson-Crick complements

```
RNA(b_1, …, b_n) {
    for k = 5, 6, …, n-1
        for i = 1, 2, …, n-k
            j = i + k
            Compute M[i, j]
                                  ↖ using recurrence
    return M[1, n]
}
```

Dynamic Dijkstra



Dynamic Dijkstra



Shortest paths: dynamic programming

Def. $OPT(i, v)$ = cost of shortest $v \to t$ path that uses $\le i$ edges.

- Case 1: Cheapest $v \to t$ path uses $\le i-1$ edges.
  - $OPT(i, v) = \mathrm{OPT}(i-1, v)$

optimal substructure property
(proof via exchange argument)

- Case 2: Cheapest $v \to t$ path uses exactly i edges.
  - if (v, w) is first edge, then OPT uses (v, w), and then selects best $w \to t$ path using $\le i-1$ edges

$$OPT(i,v) = \begin{cases} \infty & \text{if } i = 0 \\ \min\left\{ OPT(i-1,\ v),\ \min_{(v,w)\in E} \{ OPT(i-1,\ w) + c_{vw} \} \right\} & \text{otherwise} \end{cases}$$

Observation. If no negative cycles, $OPT(n-1, v)$ = cost of cheapest $v \to t$ path.

Pf. By Lemma 2, cheapest $v \to t$ path is simple. ▪

Find new flow Q

Consider the following problem. You are given a flow network with unit capacity edges: It consists of a directed graph $G = (V, E)$, a source $s \in V$, and a sink $t \in V$; and $c_e = 1$ for every $e \in E$. You are also given a parameter k. The goal is to delete k edges so as to reduce the maximum s-t flow in G by as much as possible. In other words, you should find a set of edges $F \subseteq E$ so that $|F| = k$ and the maximum s-t flow in $G = (V, E - F)$ is as small as possible subject to this.

Give a polynomial-time algorithm to solve this problem.

Find new flow solution

According to the max-flow min-cut theorem, there is an s-t cut with g edges. If g is smaller or equal to k, then we simply remove all edges in that s-t cut, disconnecting s and t, decreasing the maximum flow to 0.

If g>k, by removing k edges from the min-cut, we create a cut with value g-k.

Find new flow2

Suppose you are given a directed graph G=(V,E) ,with a positive integer capacity ce on each edge e, a source s∈V, and a sink t∈V. You are also given a maximum s-t flow in G, defined by a flow value f_e on each edge e. The flow f is acyclic: There is no cycle in G on which all edges carry positive flow. The flow f is also integer-valued.

Now suppose we pick a specific edge e* ∈ E and reduce its capacity by 1 unit. Show how to find a maximum flow in the resulting capacitated graph in time O(m + n), where m is the number of edges in G and n is the number of nodes.

Find new flow2 solution

Note first that the maximum flow in the new graph is either the same or it decreases at most by one.

This change can create an imbalance between the in-flows and out-flows at u and v respectively. To fix that we look in the residual graph and try to find a path from t to v and also a path from u to s, which can be done in O(m+n). Then we decrease the flow on all edges of the two paths from t to v and u to s. (we now have a max-flow = max-flow-1)

Look for a new augmenting path with flow 1. We can find these augmenting paths by Ford-Fulkerson algorithm which take s O(m+n) time.

Ergonomic design

Give an algorithm to decide if a given floor plan is ergonomic. The running time should be polynomial in m and n. You may assume that you have a subroutine with O(1) running time that takes two line segments as input and decides whether or not they cross in the plane.

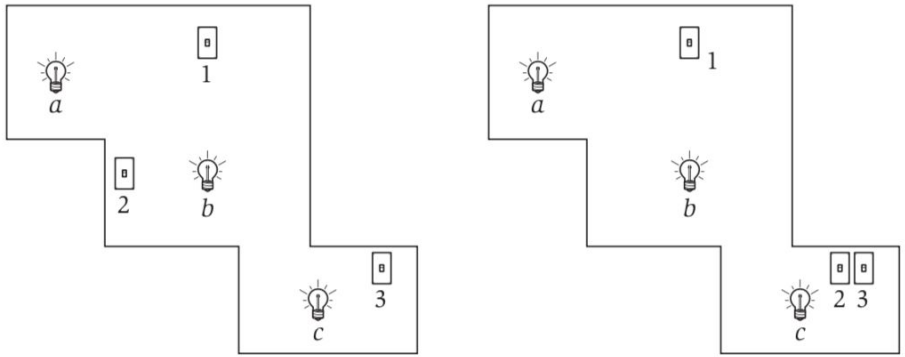


Figure 7.28 The floor plan in (a) is ergonomic, because we can wire switches to fixtures in such a way that each fixture is visible from the switch that controls it. (This can be done by wiring switch 1 to a, switch 2 to b, and switch 3 to c.) The floor plan in (b) is not ergonomic, because no such wiring is possible.

Ergonomic design

Give an algorithm to decide if a given floor plan is ergonomic. The running time should be polynomial in m and n. You may assume that you have a subroutine with O(1) running time that takes two line segments as input and decides whether or not they cross in the plane.

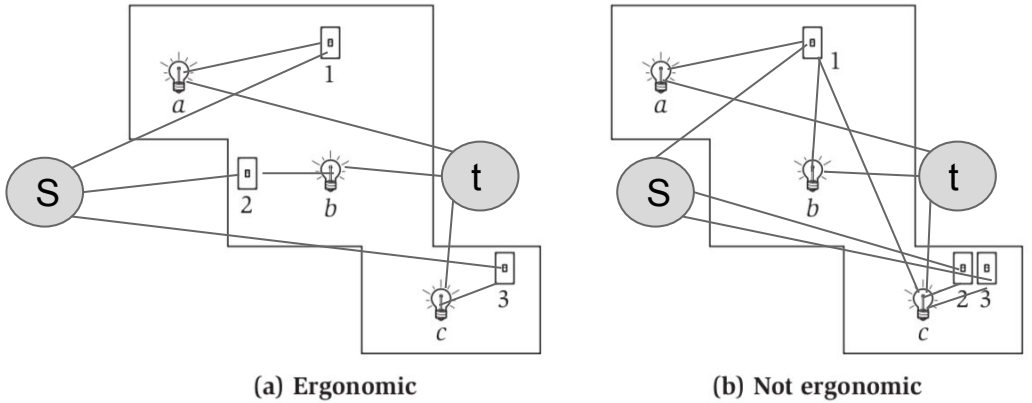


Figure 7.28 The floor plan in (a) is ergonomic, because we can wire switches to fixtures in such a way that each fixture is visible from the switch that controls it. (This can be done by wiring switch 1 to a, switch 2 to b, and switch 3 to c.) The floor plan in (b) is not ergonomic, because no such wiring is possible.

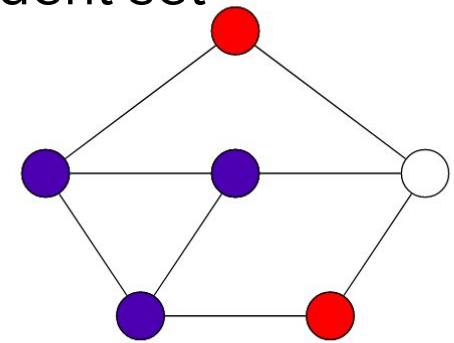
Independent Set and Clique

Given a graph G, a set of vertices V' is:

(A) An independent set: if no two vertices of V' are connected by an edge of G.

(B) Clique: every pair of vertices in V' is connected by an edge of G.

Task: reduce Clique to independent set



The **Independent Set** and **Clique** Problems

A.

To reduce a Clique Problem to an Independent Set problem for a given graph $G = (V, E)$, construct a complimentary graph $G' = (V', E')$ such that

1. $V = V'$, that is the compliment graph will have the same vertices as the original graph

2. E' is the compliment of E that is G' has all the edges that is **not** present in G

Note: Construction of the complimentary graph can be done in polynomial time

B.

1. **If there is an independent set of size k in the complement graph** G', it implies no two vertices share an edge in G' which further implies all of those vertices share an edge with all others in G forming a clique. that is **there exists a clique of size k in** G

2. **If there is a clique of size** k **in the graph** G, it implies all vertices share an edge with all others in G which further implies no two of these vertices share an edge in G' forming an Independent Set. that is **there exists an independent set of size k in** G'

Visualize

HS reduced to TSP

Hamiltonian Cycle and Traveling Salesman problems

For a given graph $G = (V, E)$, the Hamiltonian Cycle problem is to find whether G contains a Hamiltonian Cycle that is, a cycle that passes through all the vertices of the graph exactly once.

For a given weighted graph $G' = (V', E')$, with non-negative weights, and integer k' , the Traveling Salesman problem is to find whether G' contains a simple cycle of length <= k that passes through all the vertices. [Length of a cycle is the sum of weights of all the edges in the cycle].

HS reduced to TSP

To reduce the Hamiltonian Cycle Problem to the Traveling Salesman problem for a given graph $G = (V, E)$, complete the graph G, by adding edges between all pairs of vertices that were not connected in G

Let the new graph be $G' = (V', E')$ where $V' = V$ and E'={(u,v)} for any $u, v \in V'$.

For edges in G' that were also present in G , we assign a weight 0.
For other edges we assign weight 1

that is , $\forall e = (u, v) \in E'$,

$W(e) = 0$, if $(u, v) \in E$

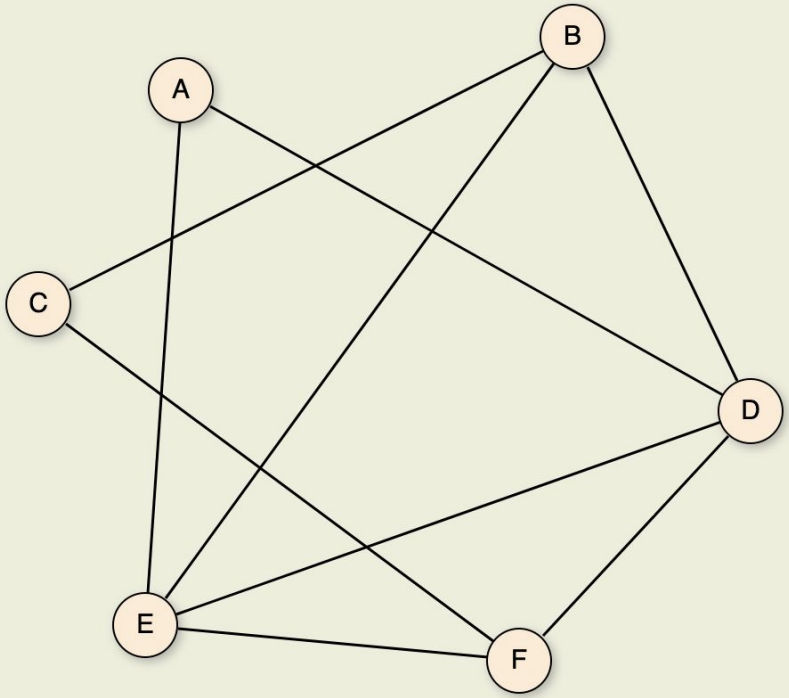
$W(e) = 1$, if $(u, v) \notin E$

.

Note: Construction of the complimentary graph can be done in polynomial time

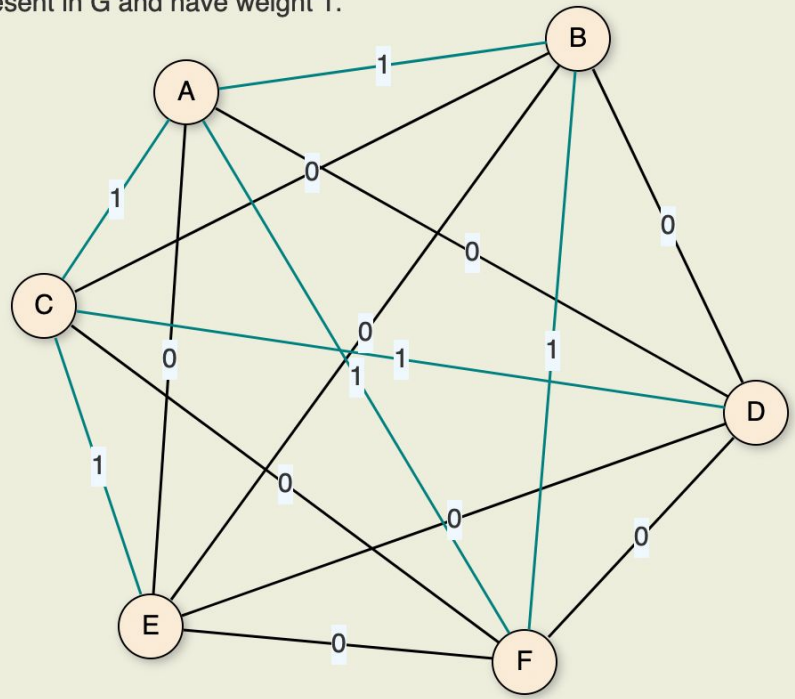
HS reduced to TSP

Let this graph G be an input to the Hamiltonian Cycle problem



HS reduced to TSP

The constructed graph G' is as below.
The blue edges were not present in G and have weight 1.



HS reduced to TSP

Hamiltonian Cycle problem reduced to an instance of Traveling Salesman Problem

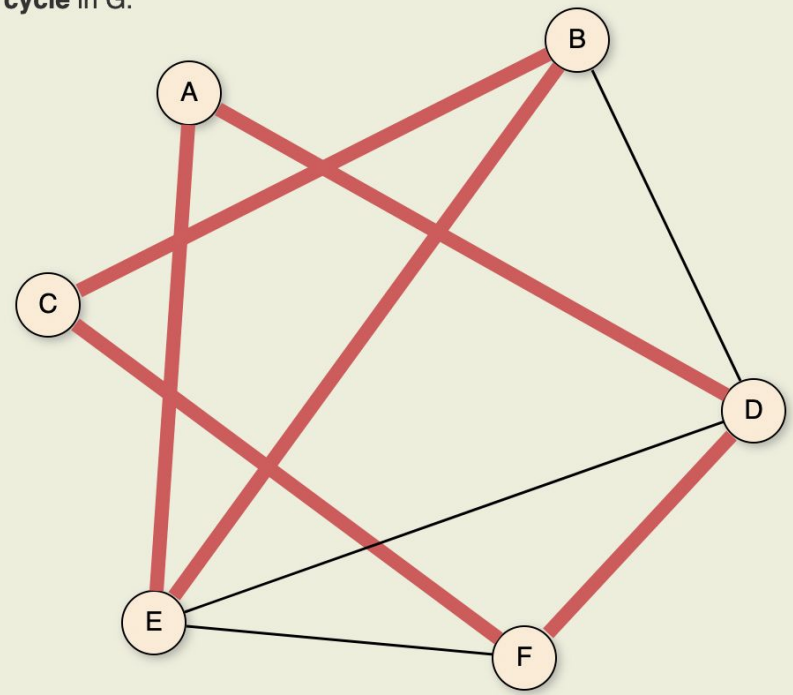
The graph G has a Hamiltonian Cycle if and only if there exists a cycle in G' passing through all vertices exactly once, and that has a length <= 0 (i.e. has a solution for the instance of Traveling Salesman Problem where $k = 0$

1. **If there is a cycle that passes through all vertice exactly once, and has length** <= 0 **in graph** G', the cycle contains only edges that were originally present in graph G. (The new edges in G' have weight 1 and hence can not be part of a cycle of length <= 0.
Hence **there exist a Hamiltonian cycle in G**

2. **If there exists a Hamiltonian Cycle in the graph** G, it forms a cycle in G' with length = 0, since a weights of all the edges is 0.
Hence **there exists a solution for Traveling Salesman Problem in** G' **with length** <= 0

HS reduced to TSP

G' has a cycle passing through all vertices exactly once with length <= 0
This cycle is a **Hamiltonian cycle** in G.



Recruiting

Suppose you're helping to organize a summer sports camp, and the following problem comes up. The camp is supposed to have at least one counselor who's skilled at each of the n sports covered by the camp (baseball, volleyball, and so on). They have received job applications from m potential counselors. For each of the n sports, there is some subset of the m applicants qualified in that sport. The question is: For a given number $k < m$, is it possible to hire at most k of the counselors and have at least one counselor qualified in each of the n sports? We'll call this the Efficient Recruiting Problem.

Show that Efficient Recruiting is NP-complete.

Stock Strategy

Given n consecutive days of a given stock, find a set of up to k non overlapping intervals, during each of which the investors buy 1,000 shares of the stock (on day bi) and then sell it (on day si). The return of a given k-shot strategy is simply the profit obtained from the m buy-sell transactions

$$1{,}000 \sum_{i=1}^{m} p(s_i) - p(b_i).$$

Your goal is to design an efficient algorithm that determines, given the sequence of prices, the k- shot strategy with the maximum possible return. your running time should be polynomial in both n and k; it should not contain k in the exponent.

Stock Strategy Solution

```python
# Find maximum profit earned from at most k stock transactions
# Input to the function are stock prices of n days and positive number k
def maxProfit(price, k):

    # get number of days n
    n = len(price)

    # profit[i][j] stores the maximum profit gained by doing
    # at most i transactions till j'th day
    profit = [[None for x in range(n)] for y in range(k + 1)]

    # fill profit matrix in bottom-up fashion
    for i in range(k + 1):
        for j in range(n):
            # profit is 0 when:
            # i = 0 i.e. for 0'th day
            # j = 0 i.e. no transaction is being performed

            if i == 0 or j == 0:
                profit[i][j] = 0
            else:
                max_so_far = 0
                for x in range(j):
                    curr_price = price[j] - price[x] + profit[i-1][x]
                    if max_so_far < curr_price:
                        max_so_far = curr_price

                profit[i][j] = max(profit[i][j-1], max_so_far)

    return profit[k][n-1]


if __name__ == '__main__':

    price = [1, 5, 2, 3, 7, 6, 4, 5]
    k = 3

    print("The maximum possible profit is", maxProfit(price, k))

```

```
T[t][i] = max(T[t][i-1], max(price[i] – price[j] + T[t-1][j]) |
                                 where j varies from 0 to i-1)
```

Interview Questions!!!

Interview Questions!!!

Interview Questions!!!

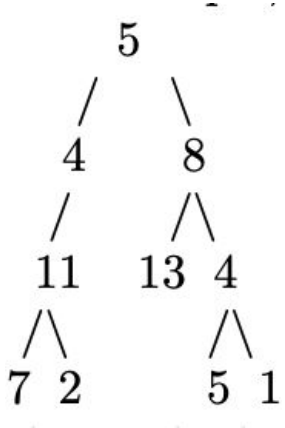
Interview Questions!!!

Interview Questions!!!

Binary tree

Given a binary tree and a sum, find all root-to-leaf paths where each path's sum equals the given sum.

For example, given the below binary tree and sum = 22,



the method returns the following: [5,4,11,2], [5,8,4,5]

Minimal height tree

a. Minimal Tree: Given a sorted (increasing order) array with unique integer elements, write an algorithm to create a binary search tree with minimal height.

Minimal height tree

```
1   TreeNode createMinimalBST(int array[]) {
2     return createMinimalBST(array, 0, array.length - 1);
3   }
4
5   TreeNode createMinimalBST(int arr[], int start, int end) {
6     if (end < start) {
7       return null;
8     }
9     int mid = (start + end) / 2;
10    TreeNode n = new TreeNode(arr[mid]);
11    n.left = createMinimalBST(arr, start, mid - 1);
12    n.right = createMinimalBST(arr, mid + 1, end);
13    return n;
```

Missing Two

You are given an array with all the numbers from 1 to N appearing exactly once, except for one number that is missing.

a. How can you find the missing number in O(N) time and 0(1) space?
b. What if there were two numbers missing?

two numbers missing

$$
\begin{aligned}
x + y &= sum \quad &&\rightarrow y = sum - x \\
x * y &= product \quad &&\rightarrow x(sum - x) = product \\
& && \quad x*sum - x^2 = product \\
& && \quad x*sum - x^2 - product = 0 \\
& && \quad -x^2 + x*sum - product = 0
\end{aligned}
$$