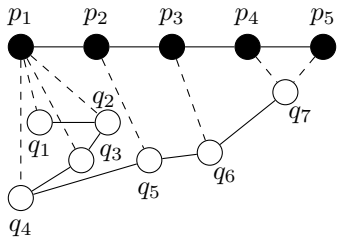
Discrete Fréchet distance Consider Professor Bille going for a walk with his personal dog. The professor follows a path of points $p_1, \ldots, p_n$ and the dog follows a path of points $q_1, \ldots, q_m$. We assume that the walk is partitioned into a number of small steps, where the professor and the dog in each step either both move from p_i tp p_{i+1} and from q_j to q_{j+1}, respectively, or only one of them moves and the other one stays.

The goal is to find the smallest possible length L of the leash, such that the professor and the dog can move from p_1 and q_1, resp., to p_n and q_n. They cannot move backwards, and we only consider the distance between points. The distance L is also known as the discrete Fréchet distance.

We let $L(i, j)$ denote the smallest possible length of the leash, such that the professor and the dog can move from p_1 and q_1 to p_i and q_j, resp. For two points p and q, let $d(p, q)$ denote the distance between the them.

In the example below the dotted lines denote where Professor Bille (black nodes) and the dog (white nodes) are at time 1 to 8. The minimum leash length is $L = d(p_1, q_4)$.



Q1: Recurrence Give a recursive formula for $L(i, j)$.

Solution

$$L(i,j) = \begin{cases} d(p_1, q_1) & \text{for } i = 1 \wedge j = 1 \text{ (1)} \\ \max(d(p_1, q_j), L(1, j-1)) & \text{for } i = 1 \text{ (2)} \\ \max(d(p_i, q_1), L(i-1, j)) & \text{for } j = 1 \text{ (3)} \\ \max(d(p_i, q_j), \min\{L(i-1, j), L(i, j-1), L(i-1, j-1)\}) & \text{otherwise (4)} \end{cases}$$

The three first cases correspond to the following three base cases: (1) both Professor Bille and the dog are on the first nodes in their respective paths (p_1/q_1) (2) Professor Bille stands on the first node (p_1) while only the dog is moving (3) the dog stands on the first node (q_1) and Professor Bille is the only who is moving.

In the general case (4), we must use a leash of length at least the distance between p_i and q_j since they could not otherwise stand on p_i and q_i respectively. Furthermore the leash must be long enough for the steps made so far. We consider the following three cases for the previous step and take the best of these (ie. the smallest): (a) Professor Bille moves (b) the dog moves (c) they both move.

Q2: Algorithm Give an algorithm that computes the length of the shortest possible leash. Analyze space and time usage of your solution.

```
COMPUTE-LENGTH()
   L := n × m table

   for i = 1 to n do
      for j = 1 to m do
```

```
        if i = 1 and j = 1 then
           L[i, j] := d(p_1, q_1)
        else if i = 1 then
           L[i, j] := max(d(p_1, q_j), L[1, j - 1])
        else if j = 1 then
           L[i, j] := max(d(p_i, q_1), L[i - 1, 1])
        else
           L[i, j] := max(d(p_i, q_j), min(L[i - 1, j], L[i, j - 1], L[i - 1, j - 1]))
        end if
     end for
  end for

  return L[n, m]
```

Since we store the results to the subproblems in a table L of size $n \times m$ we use $\Theta(nm)$ space. The outer loop always make n iterations and the inner loop always makes m iterations per iteration of the outer loop. All other operations can be done in constant time, and thus the total time of the algorithm is $\Theta(nm)$.

Q3: Print solution Extend your algorithm to print out paths for the professor and the dog. The algorithm must return where the professor and the dog is at each time step. Analyze the time and space usage of your solution.

Solution First use the above algorithm to compute L and then call PRINT-ROUTE(n, m) to print the sequence of locations the Professor and the dog will visit.

```
PRINT-ROUTE(i, j)
  if i > 1 and L[i - 1, j] <= L[n,m] then
     PRINT-ROUTE(i - 1, j)
  else if j > 1 and L[i, j - 1] <= L[n, m] then
     PRINT-ROUTE(i, j - 1)
  else if i > 1 and j > 1
     PRINT-ROUTE(i - 1, j - 1)

  print (i, j)
```

We do not need to save any additional things to reconstrunct the paths, thus the space usage is still $\Theta(nm)$. A path from (p_1, q_1) to (p_n, q_m) can contain at most $n+m$ nodes as at least one must move in each step and they cannot move backwards. Thus PRINT-ROUTE will call itself less than $n+m$ times. All the operations in PRINT-ROUTE (except the recursive calls) can be done in constant time. Thus the running time of PRINT-ROUTE is $O(n+m)$. The overall running time is therefore $\Theta(nm) + O(n+m) = \Theta(nm)$.

Blood donors At the halloween party at a well-known academic institution north of Copenhagen not all went smooth and some students had to be taken to medical emergency treatment at *Rigshospitalet*. In total 150 had to get a transfusion of one bag of blood. The hospital had 155 bags in stock. The distribution of blood groups in the supply and amongst the students is shown in the table below.

Blood type	A	B	0	AB
Bags in stock	44	31	42	38
Demand	37	33	40	40

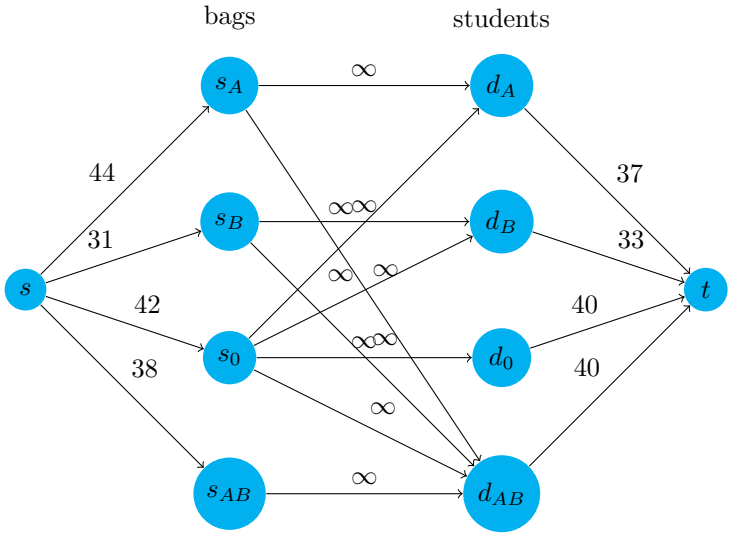
Type **A** patients can only receive blood of type **A** or type **0**; type **B** patients can receive only type **B** or type **0**; type **0** patients can receive only type **0**; and type **AB** patients can receive any of the four types.

Model the problem as a flow network problem. Draw the corresponding network, and interpret the meaning of the nodes, and edges (edges capacities). Describe how to check whether every student can get a transfusion, otherwise how many can get one.

You do not have to solve the problem explicitly.

Solution The problem can be modelled as a flow problem. Create a graph G with a source node s, a sink node t, 4 nodes for the supply of each blood type s_T (where T is the blood type), and 4 nodes for the demand of each blood type d_T. Make an edge from s to s_T with capacity equal to the number of bags in stock of blood type T. Make an edge from s_T to d_U with unlimited capacity iff a patient with blood type T can use blood of type U. Finally make edges from d_T to t with capacity equal to the number of students with blood type T.

Use a maximum flow algorithm like Ford-Fulkerson to find the maximum $s-t$ flow in g. If the value of the maximum flow equals the number of students then all students can get a transfusion, otherwise the maximum flow is the number of students that can.



]

Correctness.

First we show a flow in G corresponds to a valid assignment of bags to students. Since all the capacities are integral, we can assume the flow on any edge is integral. Let a flow of $f(s_T, d_U)$ units from s_T to d_U correspond to $f(s_T, d_U)$ bags of bloodtype T is given to students with bloodtype U. Because we only have edges from s_T to d_U if students with blood type T can use bags of type U, we only assign bags to students whose blood type matches. Because of the capacity constraints on the edges from s to s_T and flow conservation in s_T we do not use more bags than we have of any bloodtype. Because of the capacity

constraints on the edges from d_T to t and flow conservation in d_T, we do not give blood to more students than needs it.

Secondly, we show there exists a valid flow in G for any valid assignment of bags to students. We construct the flow as follows:

$$f(u,v) = \begin{cases} \#\text{ bags of type T used} & \text{for } u = s \wedge v = s_T \\ \#\text{ bags of type T assigned to students of type D} & \text{for } u = s_T \wedge v = u_D \\ \#\text{ students with type T satisfied} & \text{for } u = d_T \wedge v = t \end{cases}$$

The capacity constraints are satisfied for edges from s to s_T because an assignment of bags cannot use more bags than we have and the capacity on the edges are exactly the number of bags available of that type. A student can be assigned at most one bag of blood, and thus the capacity constraints on edges from d_T to t are satisfied since the capacity is the number of students with the blood type.
Flow conservation in the nodes s_T is satisfied since the number of bags of type T used is equal to the number of students they are given to. There is flow conservation in the nodes d_T as the number of students with blood type T who is given blood is equal to the sum of the different types of blood they are given.

Combined this means a maximum flow corresponds to a maximum number of students can get a transfusion.

Lexicographically smallest shift In chemical databases for circular molecules, each molecule is represented by a circular string of chemical characters. To allow faster lookup and comparisons of molecules, one wants to store each circular string by a canonical linear string. A natural choice for a canonical in ear strings the one that is lexicographically smallest. That gives the following computational problem.

Assume we are given a string $T = x_1 \dots x_n$ of length n. A *shift* of T by s, $0 \leq s < n$, is the string $T^s = x_{s+1}x_{s+2}\dots x_n x_1 x_2 \dots x_s$. In this problem we want to find the *lexicographically smallest shift*, i.e. the shift s where T^s is lexicographically smallest among $T^0, \dots, T^{n-1}$. Eg. $T^2 = T^7 = \texttt{aababaabab}$ are the lexicographically smallest shifts of the string

$$T = \texttt{abaababaab}$$

Q1 State all s where T^s is a lexicographically smallest shift of the string

$$T = \texttt{bcabaabcabaabcabaa}$$

Solution $T^4 = T^{10} = T^{16} = \texttt{aabcabaabcabaabcab}$

Q2 Describe an algorithm that given a string T of length n over an alphabet of size $O(1)$ computes all s where T^s is a lexicographically smallest shift of T. State the algorithms running time.

Solution Construct the suffix tree S of the string $TT\$$ where \$ is a new character lexicographically larger than all others in the alphabet. Assume all the children lists in S are lexicographically ordered from left to right. Follow the left-most path in S from the root. Stop when a node v where there is at least $n+2$ characters on the path from the root to it is reached. Look at all the leafs in the subtree of v, and output the indices of the suffixes they correspond to.

Correctness. First we must argue, that the algorithm always can find a node v such that the number of characters from the root to v is at least $n+2$ by only looking at the left-most path. Assume this is not the case, then the left-most leaf in the tree must correspond to a suffix $s\$$ of length less than $n+2$. Because

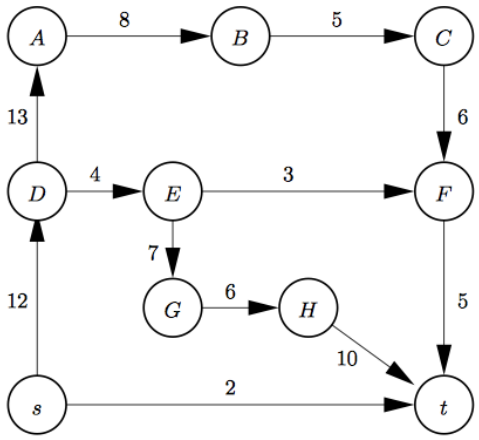
we have repeated T after itself, then there must exist another suffix $sT\$$. But $s\$$ is lexicographically larger than $sT\$$ (because of how we chose \$), which contradicts with all children being lexicographically ordered from left to right.

All the suffixes represented by leafs in the subtree under v share the same first n characters, so if we only look at the n first characters in each suffix they must be lexicographically the same, thus they all correspond to lexicographically smallest shifts.

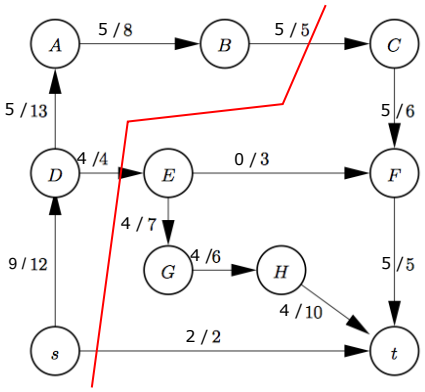
Analysis. Constructing the suffix tree can be done in $O(2n+1) = O(n)$ time since the length of the string is $2n+1$ and the alphabet size is constant. A simple path from the root to any node is at most n nodes, so following the left-most path takes $O(n)$ time. Reporting the indices from all the leafs below v takes $O(n)$ time since we must traverse a tree with less than n nodes. So the total running time is $O(n)$.

Maximum flow and Minimum Cut Consider the below network with edge capacities.

Give a maximum flow from s to t in the network (state for each edge the flow along the edge), state the value of the maximal flow, and a minimum cut where the capacity of the flow equals the maximal flow.



Solution



The maximum flow is 11. A minimum s-t cut is $\{s, D, A, B\}$ / $\{C, E, G, H, F, t\}$ and is illustrated by the red line. (Another minimum s-t cut is $\{s, D, A, B, C, F\}$ / $\{E, G, H, t\}$ - answering just one of these is sufficient)

The Edmonds-Karp algorithm Consider the Edmonds-Karp algorithm applied to the above graph to find a maximal flow. State the augmenting paths found by the algorithm. State for each augmenting path the nodes on the path and the value augmented with along the path.

Solution

augmenting path	value
$s \to t$	2
$s \to D \to E \to F \to t$	3
$s \to D \to E \to G \to H \to t$	1
$s \to D \to A \to B \to C \to F \to t$	2
$s \to D \to A \to B \to C \to F \to E \to G \to H \to t$	3