CAL solved exercises – Exams

Diogo Miguel Ferreira Rodrigues (dmfrodrigues2000@gmail.com)

Diogo Miguel Ferreira Rodrigues (dmfrodrigues2000@gmail.com)

Contents

Exam normal 2018/19

19N-P1 Part 1

Question 1

With dynamic programming, the function ${}^nC_k$ below can be calculated in how much time ($T(n,k)$) and space ($S(n,k)$)? (choose the lowest valid limits)

$$ {}^nC_k = \begin{cases} 1 & : k = 0 \\ 1 & : k = n \\ {}^{n-1}C_k + {}^{n-1}C_{k-1} & : \text{otherwise} \end{cases} $$

A) $T(n,k) = S(n,k) = O(1)$

B) $T(n,k) = O(k(n-k))$, $S(n,k) = O(n-k)$ ✓

C) $T(n,k) = O(n-k)$, $S(n,k) = O(k(n-k))$

D) $T(n,k) = S(n,k) = O(n+k)$

To compute ${}^{10}C_3$ we need to compute all values in bold in the table:

						k				
n	0	1	2	3	4	5	6	7	8	9	10
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					
6	**1**	**6**	**15**	**20**	15	6	1				
7	**1**	**7**	**21**	**35**	35	21	7	1			
8	1	**8**	**28**	**56**	70	56	28	8	1		
9	1	9	**36**	**84**	126	126	84	36	9	1	
10	1	10	45	**120**	200	252	200	120	45	10	1

This example is enough to understand that, to compute ${}^nC_k$, one has to compute $k+1$ columns of the pascal triangle, each being $n-k+1$ rows tall. Given ${}^{n-1}C_k$ and ${}^{n-1}C_{k-1}$ have already been computed, computing ${}^nC_k$ takes time $O(1)$, so in total we require $O((k+1)(n-k+1)) = O(k(n-k))$ time.

To calculate all the needed values of column k' we only need to keep the $n-k+1$ values from the previous column $k'-1$ and an additional array of size $n-k+1$ to store the values of the new column k' the values from row $n'-1$. Therefore we need $2*(n-k+1)$ integers, which means we need $O(n-k)$ memory.

Question 2

The *quicksort* algorithm has a partition step, where the given array is rearranged into two subarrays with values less and greater than a *pivot*, followed by a recursive step where the two subarrays are sorted by the same method. What type of algorithm is this?

A) Dynamic programming

B) Greedy

C) Backtracking

D) **Divide-and-conquer** ✓

Question 3

An algorithm that explores a state space, searching for a solution state, is essentially an algorithm of which kind?

A) **Dynamic programming** ✓

B) Greedy

C) Backtracking

D) Divide-and-conquer

Question 4

The following algorithm (coded in C++) determines if an array a of size n has a subset with sum x. What kind of algorithm is this?

```
1 bool hasSum(int a[], int n, int x){
2     if(x == 0) return true;
3     else if(n == 0) return false;
4     else return hasSum(a, n-1, x-a[n-1]) ||
5                 hasSum(a, n-1, x);
6 }
```

A) Dynamic programming

B) Greedy

C) **Backtracking** ✓

D) Divide-and-conquer

Question 5

Which of the following properties is not associated to the cycle invariant in an iterative algorithm?

A) at the beginning of the cycle, being implied by the pre-condition

B) at the end of the cycle, to imply the post-condition

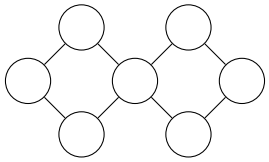
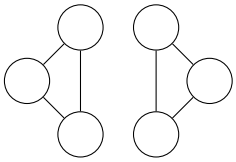
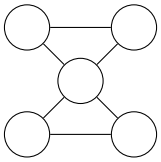
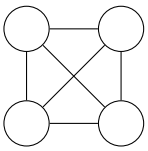
C) to be true at each cycle iteration

D) **to be false in the last cycle iteration** ✓

Diogo Miguel Ferreira Rodrigues (dmfrodrigues2000@gmail.com)

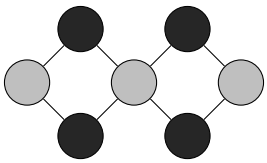
Question 6

Which of the following graphs is bipartite?

A)  ✓ B)  C)  D) 

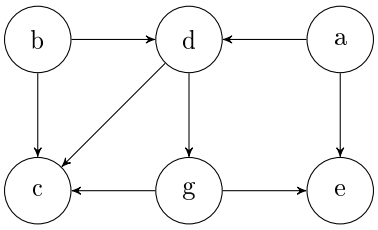
A bipartite graph is a graph whose nodes can be split into two sets where two nodes in the same set are not adjacent; this is equivalent to say the graph can be coloured with two colours in a graph colouring problem.

If we colour the graph from option A) we can easily see why it is bipartite:



Question 7

Which of the following options is not a possible topological order of vertices of the following graph?



A) a-b-d-g-c-e

B) a-b-d-g-e-c

C) **a-d-c-g-e-b** ✓

D) b-a-d-g-e-c

Since $b \rightsquigarrow d$, b must be before d, which is not the case in option C)

Question 8

Distances through the shortest path of a vertex s to each vertex in a weighted directed acyclic graph $G = (V, E)$, with possible negative-weight edges, can be calculated in what time?

A) $O(|V| + |E|)$ ✓

B) $O((|V| + |E|) \times \log |V|)$

C) $O(|V| \times |E|)$

D) $O(|V|^3)$

This is a shortest path problem to all vertices in a DAG, which can be solved first by sorting vertices by topological order in $O(|V| + |E|)$ and then iterate over the vertices in topological order and update distances in $O(|V| + |E|)$.

Question 9

A graph $G = (V, E)$ can be considered dense when the relation between the number of vertices and the number of edges is of what order of magnitude?

A) $|V| - |E|$

B) $|V|^2 - |E|$ ✓

C) $|V| - |E|^2$

D) $2 \times |V| - |E|$

A complete simple graph has $|E| = \frac{1}{2}|V|(|V| - 1)$, so the graph is considered dense when $|E| \approx |V|^2$.

Question 10

Which shortest path algorithms in a graph $G = (V, E)$ use dynamic programming?

A) A*, Dijkstra

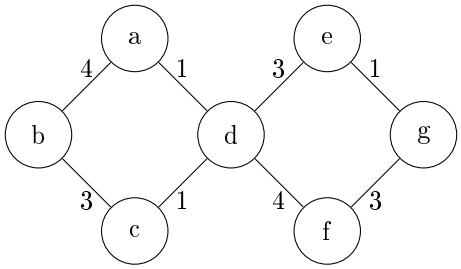
B) **Bellman-Ford, Floyd-Warshall** ✓

C) Bellman-Ford, A*

D) Floyd-Warshall, D'Esopo-Pape

Question 11

What is the weight of the minimum spanning tree of the following graph?

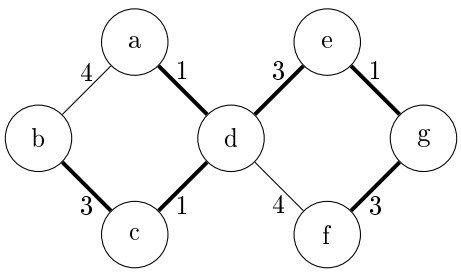


A) 6

B) 9

C) **12** ✓

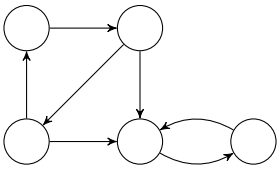
D) 18



Diogo Miguel Ferreira Rodrigues (dmfrodrigues2000@gmail.com)

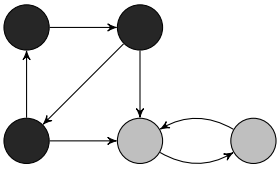
Question 12

How many strongly connected components does the following graph have?



A) 1
B) **2** ✓
C) 3
D) 0

The two SCCs are coloured dark and light gray.



Question 13

Which of the following graphs is not envolved in Ford-Fulkerson algorithm to compute maximum flow in a transport network?

A) **paths graph** ✓
B) capacities graph
C) flows graph
D) residuals graph

The Ford-Fulkerson algorithm uses three data structures:

- Capacities graph – input graph.
- Flows graph – output graph.
- Residuals graph – auxiliary structure, can be updated from the capacities graph if we do not intend to reuse it.

Question 14

About a cut (S, T) in a transport network represented by a graph $G = (V, E)$, which of the following options is false?

A) The capacity of cut (S, T) is the sum of capacities of cutten edges from S to T.
B) (S, T) is a partition of V into two sets, S and $T = V - S$ such that the source is in S and the sink is in T.
C) **The minimum cost in a transport network is equal to the capacity of the minimum cut.** ✓
D) The maximum flow in a transport network is equal to the capacity of the minimum cut.

A) follows from the definition of cut capacity.

B) is one of the necessary conditions for (S, T) to be a cut.

C) states the opposite of the *max-flow min-cost theorem.*

D) follows from the *max-flow min-cost theorem.*

Question 15

A path visiting each edge of a graph exactly once is called a _____ path.

A) **eulerian** ✓
B) hamiltonian
C) chinese postman
D) travelling salesman

Question 16

The Knuth-Morris-Pratt algorithm has the objective of:

A) **Searching exact matches of a pattern in a text.** ✓
B) Searching approximate matches of a pattern in a text.
C) Generating the finite automata corresponding to the pattern one wants to search.
D) Pre-processing the pattern so as to perform advances the same size as the pattern.

Question 17

How much time does it take to compute the edit distance between a pattern P and a text T?

A) $O(|P| \times |T|)$ ✓
B) $O(|P| + |T|)$
C) $O(|P| \times \log |T|)$
D) $O(|P|^2 \times |T|)$

Question 18

About the Gale-Shapley algorithm for stable marriages considering complete preferences, it is true that:

A) Its time complexity does not depend on the size of the groups to be paired.
B) **The choice of proposing group affects the result.** ✓
C) The choice of proposing group affects the result stability.
D) The choice of proposing group affects the cycle variant.

A) is false, since its time complexity is $O(n^2)$ where n is the size of each set.

B) is true, because there can be more than one optimal matching.

C) is false, because this algorithm always finds a stable matching.

D) is false, because it does not affect the cycle variant, which is $I = n - i$ where i is the cycle variable.

Question 19

Which of the following problems has an efficient solution?

A) Hamiltonian path
B) Vertex cover
C) Travelling salesman
D) **Topological sort** ✓

Question 20

To prove a problem X is NP-complete, considering a known NP-complete problem Y, it is necessary to:

A) Reduce Y to X, in constant time
B) Reduce X to Y, in polynomial time
C) Reduce Y to X, in sub-exponential time
D) **Show X is in NP** ✓

19N-P2 Part 2

Question 1

Consider the binary square matrix M, with 2 dimensions and variable height/width, as a simplified representation of the map of a certain region, where 1 represents land and 0 represents water. We want to develop procedures to obtain information about the islands in the region, represented in the matrix as adjacent 1's. In the example below, M represents a region made of two islands.

$$M = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Solutions to the following items should be presented in C++ or pseudocode.

Item a

Implement an algorithm to determine the number of islands in the map, using one (or more) of the following approaches: greedy, backtrack and divide-and-conquer. Identify the followed approaches as well as their time complexities, justifying [note: for the example matrix, the result is 2].

```
#include <bits/stdc++.h>
using namespace std;

vector< vector<uint8_t> > M, visited;
void dfs(int h, int w){
    if(visited[h][w] || !M[h][w]) return;
    const size_t &H = M.size(), &W = M[0].size();
    visited[h][w] = true;
    if(h-1 >= 0) dfs(h-1, w);
    if(h+1 <  H) dfs(h+1, w);
    if(w-1 >= 0) dfs(h, w-1);
    if(w+1 <  W) dfs(h, w+1);
}

int main(){
    int H, W; cin >> H >> W;
    M = vector< vector<uint8_t> >(H, vector<uint8_t>(W));
    visited = vector< vector<uint8_t> >(H, vector<uint8_t>(W, false));
    for(int h = 0; h < H; ++h) for(int w = 0; w < W; ++w) cin >> M[h][w];

    int num_islands = 0;
    for(int h = 0; h < H; ++h){
        for(int w = 0; w < W; ++w){
            if(M[h][w] && !visited[h][w]){
                ++num_islands;
                dfs(h, w);
            }
        }
    }

    cout << num_islands << endl;
    return 0;
}
```

We used the backtracking approach through a depth-first search. The time complexity of this algorithm is $O(N)$, since each cell is set as visited only once, and its value is checked at most 5 times (from the 4 sides the DFS might eventually want to know if it has already been visited, plus once by the main cycle that checks if it has already been visited or not).

Item b

Implement an algorithm to determine the area of the largest island (number of cells of the largest island), using dynamic programming. Mention and justify its time complexity [node: for the example matrix, the result is 3].

```cpp
#include <bits/stdc++.h>
using namespace std;

vector< vector<uint8_t> > M, visited;

int dfs(int h, int w){
    if(visited[h][w] || !M[h][w]) return 0;
    const size_t &H = M.size(), &W = M[0].size();
    visited[h][w] = true;
    int ret = 1;
    if(h-1 >= 0) ret += dfs(h-1, w);
    if(h+1 <  H) ret += dfs(h+1, w);
    if(w-1 >= 0) ret += dfs(h, w-1);
    if(w+1 <  W) ret += dfs(h, w+1);
    return ret;
}

int main(){
    int H, W; cin >> H >> W;
    M = vector< vector<uint8_t> >(H, vector<uint8_t>(W));
    visited = vector< vector<uint8_t> >(H, vector<uint8_t>(W, false));
    for(int h = 0; h < H; ++h) for(int w = 0; w < W; ++w) cin >> M[h][w];

    int max_island_area = 0;
    for(int h = 0; h < H; ++h){
        for(int w = 0; w < W; ++w){
            max_island_area = max(max_island_area, dfs(h, w));
        }
    }

    cout << max_island_area << endl;
    return 0;
}
```

I used a backtracking approach, which probably does not qualify as dynamic programming. Since I have not found any dynamic programming approach, suggestions are welcome.

Question 2

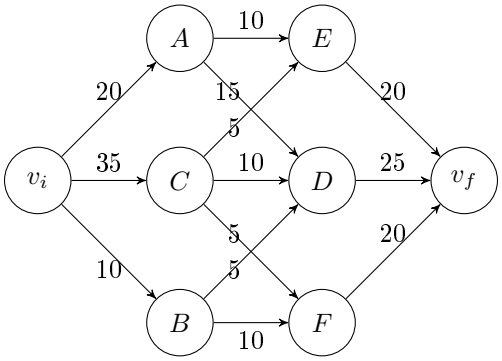
Consider three storage units A, B and C, having 20 t, 10 t and 35 t of a certain product respectively. We intend to deliver to three destinations D, E and F the amounts of 25 t, 20 t and 20 t of product respectively. The availabilities of transport, by truck, between different points are the following:

Origin \ Destin.	D	E	F
A	15	10	-
B	5	-	10
C	10	5	5

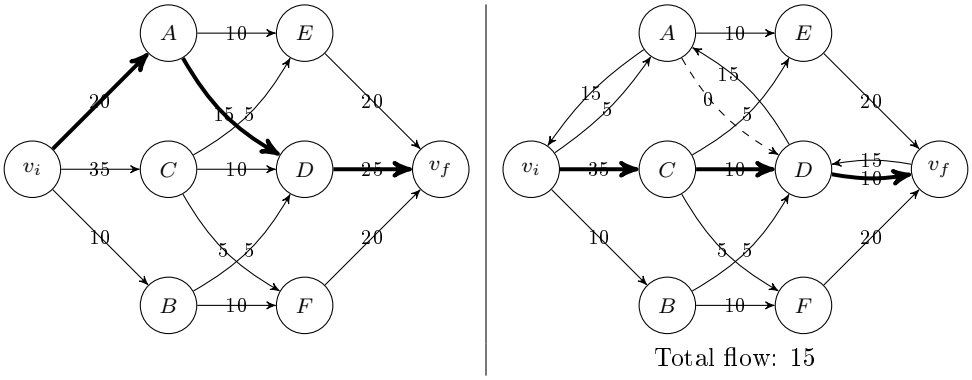
Item a

Find the best transportation plan, indicating the maximum network flow. Suggestion: consider an initial node aggregating supply and another aggregating demand.

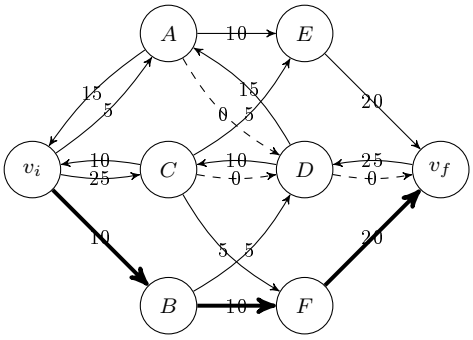
The following graph represents the flow graph between storage units and destinations.



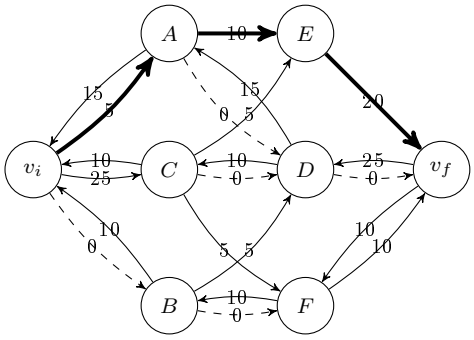
We will use the Edmonds-Karp algorithm, which consists of selecting the shortest augmenting path (in case of tie, we will choose the highest-capacity shortest augmenting path).



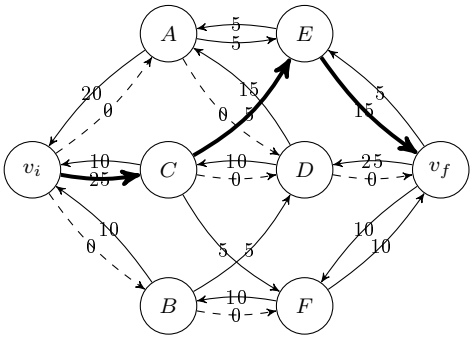
Diogo Miguel Ferreira Rodrigues (dmfrodrigues2000@gmail.com)



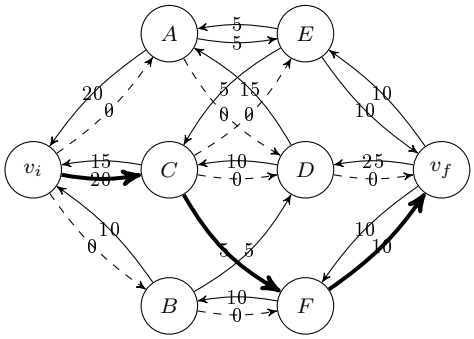
Total flow: 25



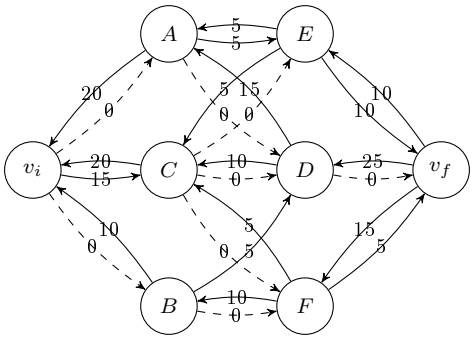
Total flow: 35



Total flow: 40



Total flow: 45



Total flow: 50

The optimal transportation plan is:

$$
\begin{aligned}
A &\rightarrow D \ (+15)\\
C &\rightarrow D \ (+10)\\
B &\rightarrow F \ (+10)\\
A &\rightarrow E \ \ (+5)\\
C &\rightarrow E \ \ (+5)\\
C &\rightarrow F \ \ (+5)
\end{aligned}
$$

which allows transporting 50 t from storage units to destinations.

Item b

Check if all destinations are supplied in the current network configuration. If not, point out what changes would have to be made to ensure the complete supplying of all destinations.

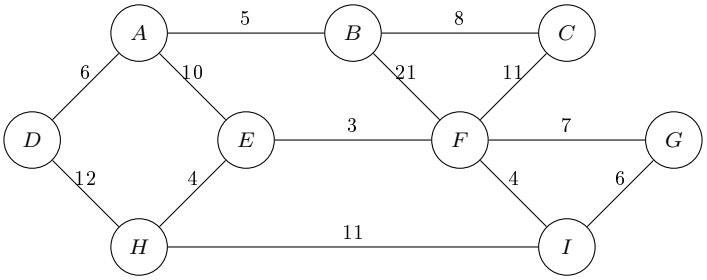
Diogo Miguel Ferreira Rodrigues (dmfrodrigues2000@gmail.com)

Not all destinations are completely supplied; the destinations that are not completely supplied depend on the algorithm we choose to calculate the maximum flow, but according to the algorithm we used destinations E and F are not completely supplied, as they demanded more 10 t and 5 t respectively.

To guarantee complete supplying of all destinations, we would have to add more 10 t transport capacity in $C \to E$, and more 5 ton in $C \to F$.

Item c

Consider the following undirected graph that represents a map, where vertices represent storage units in a city and the edge values are distances between adjacent units. Find the shortest path between deposits A and G, mentioning the algorithm you used and showing all intermediate steps of your solution.



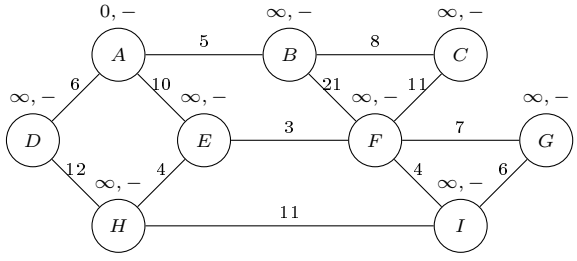
I will use Dijkstra's algorithm.

Algorithm 1 Dijkstra's algorithm

```
1:  function DIJKSTRA(G(V, E), s)
2:      for v ∈ V do dist(v) ← ∞, prev(v) ← NULL
3:      Q ← V
4:      dist(s) ← 0
5:      while |Q| > 0 do
6:          u ← vertex of Q with least dist(u)
7:          Q ← Q\{u}
8:          for v ∈ ADJ(G, u) do
9:              c' ← dist(u) + w(u, v)
10:             if c' < dist(v) then
11:                 dist(v) ← c', prev(v) ← u
12:     return dist, prev
```

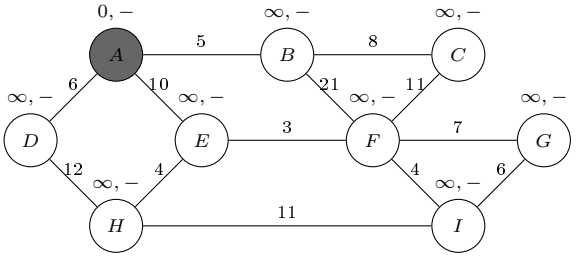
Below I present a trace of the program. In parenthesis is *dist* and *prev* for each node; nodes in light gray are those not in Q; nodes in dark gray are active nodes (current value of u); nodes with double line saw their distance updated.

Line 5

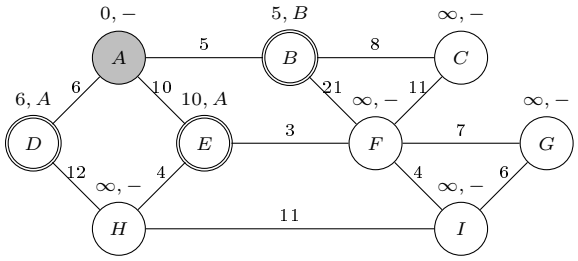


We start by setting $dist(A) \leftarrow 0$.

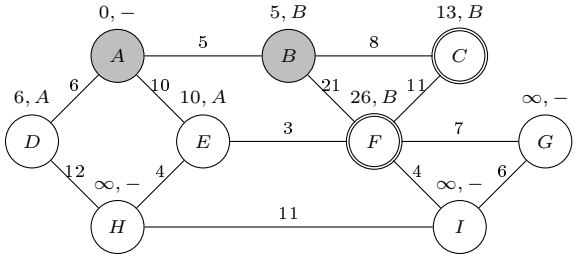
Line 8



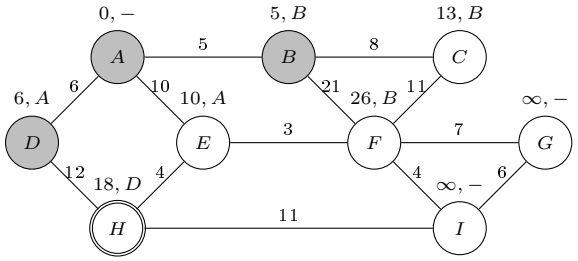
Select node in Q with least *dist*, which is A.



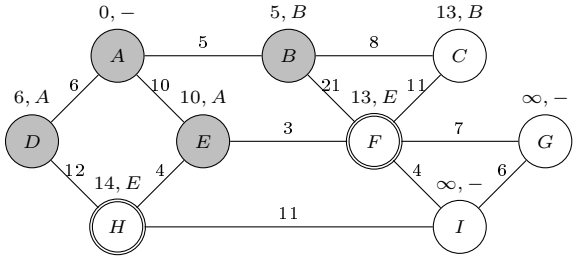
B, D and E are adjacent to A.
$dist(A) + w(A, B) = 5 < \infty$ so B was updated.
$dist(A) + w(A, D) = 6 < \infty$ so D was updated.
$dist(A) + w(A, E) = 10 < \infty$ so E was updated.



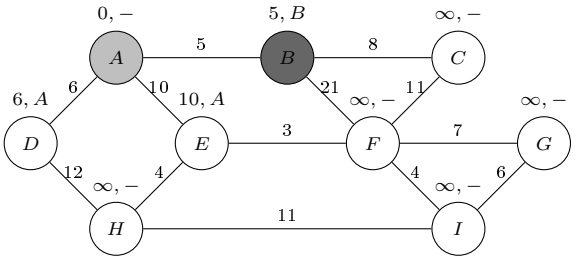
A, C and F are adjacent to B.
$A \notin Q$ so we can ignore it.
$dist(B) + w(B, C) = 13 < \infty$ so C was updated.
$dist(B) + w(B, F) = 26 < \infty$ so F was updated.



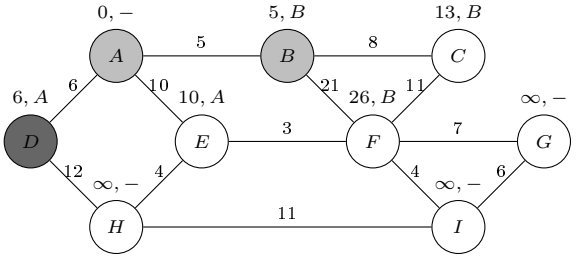
A and H are adjacent to D.
$A \notin Q$ so we can ignore it.
$dist(D) + w(D, H) = 18 < \infty$ so H was updated.



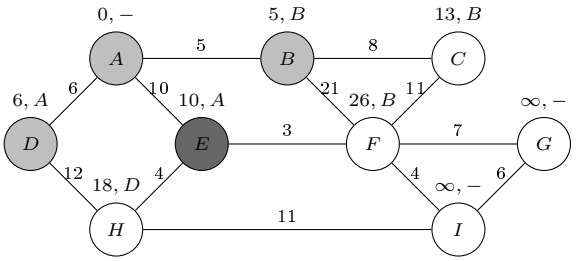
A, F and H are adjacent to E.
$A \notin Q$ so we can ignore it.
$dist(E) + w(E, F) = 13 < 26$ so F was updated.
$dist(E) + w(E, H) = 14 < 18$ so H was updated.



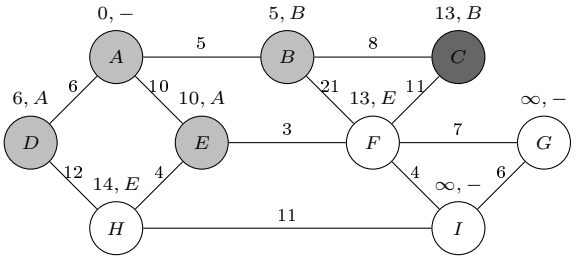
Select node in Q with least $dist$, which is B.



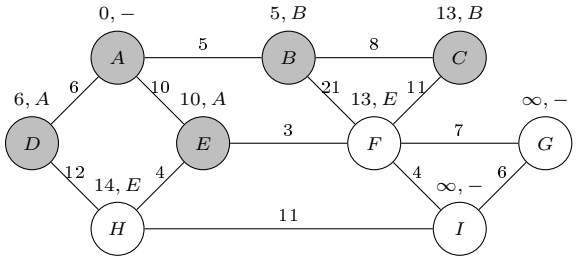
Select node in Q with least $dist$, which is D.



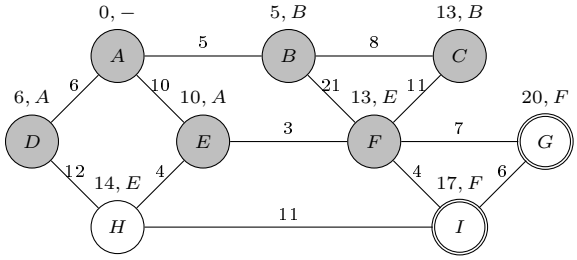
Select node in Q with least $dist$, which is E.



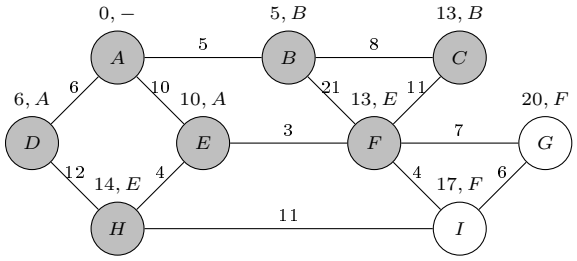
Select node in Q with least $dist$, which is C.



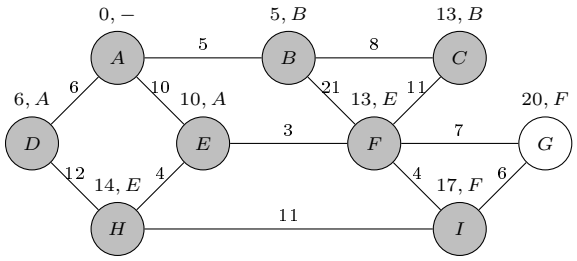
B and F are adjacent to C.
$B \notin Q$ so we can ignore it.
$dist(C)+w(C,F) = 24 \not< 13$ so F wasn't updated.



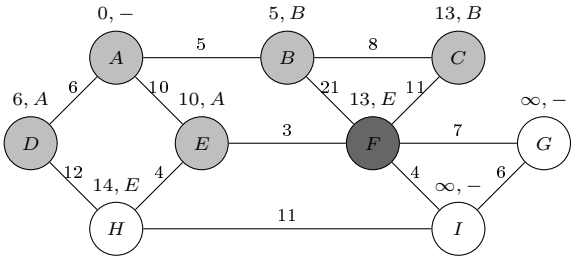
B, C, E, G and I are adjacent to F.
$B, C, E \notin Q$ so we can ignore them.
$dist(F) + w(F,G) = 20 < \infty$ so G was updated.
$dist(F) + w(F,I) = 17 < \infty$ so I was updated.



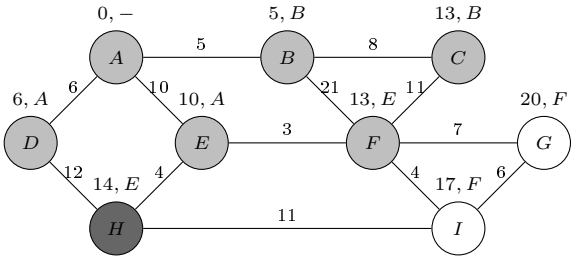
D, E and I are adjacent to H.
$D, E \notin Q$ so we can ignore them.
$dist(H)+w(H,I) = 25 \not< 17$ so I wasn't updated.



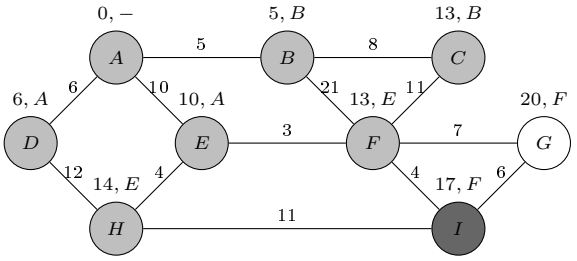
F, G and H are adjacent to I.
$F, H \notin Q$ so we can ignore them.
$dist(I) + w(I,G) = 23 \not< 20$ so G wasn't updated.



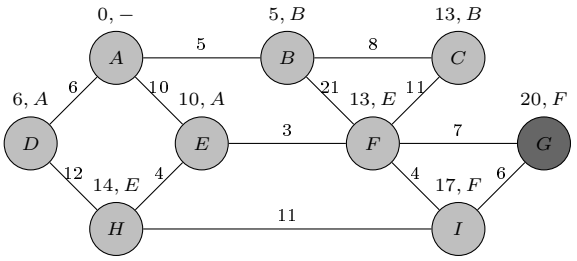
Select node in Q with least $dist$, which is F.



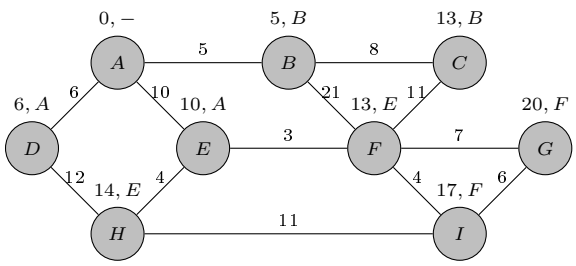
Select node in Q with least $dist$, which is H.



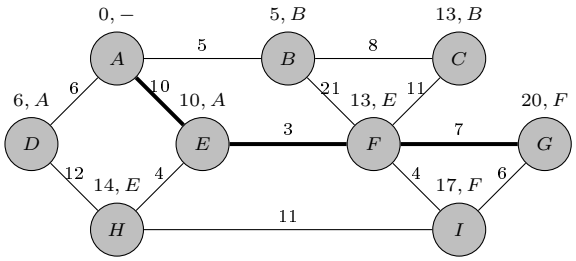
Select node in Q with least $dist$, which is I.



Select node in Q with least $dist$, which is G.



F and I are adjacent to G.
$F, I \notin Q$ so we can ignore them.



The shortest path from A to G is $A \to E \to F \to G$, with a total weight of 20.

Question 3

Consider the following 18-characters string, and answer with justification to all the following items.

"bananas e ananases"

Item a

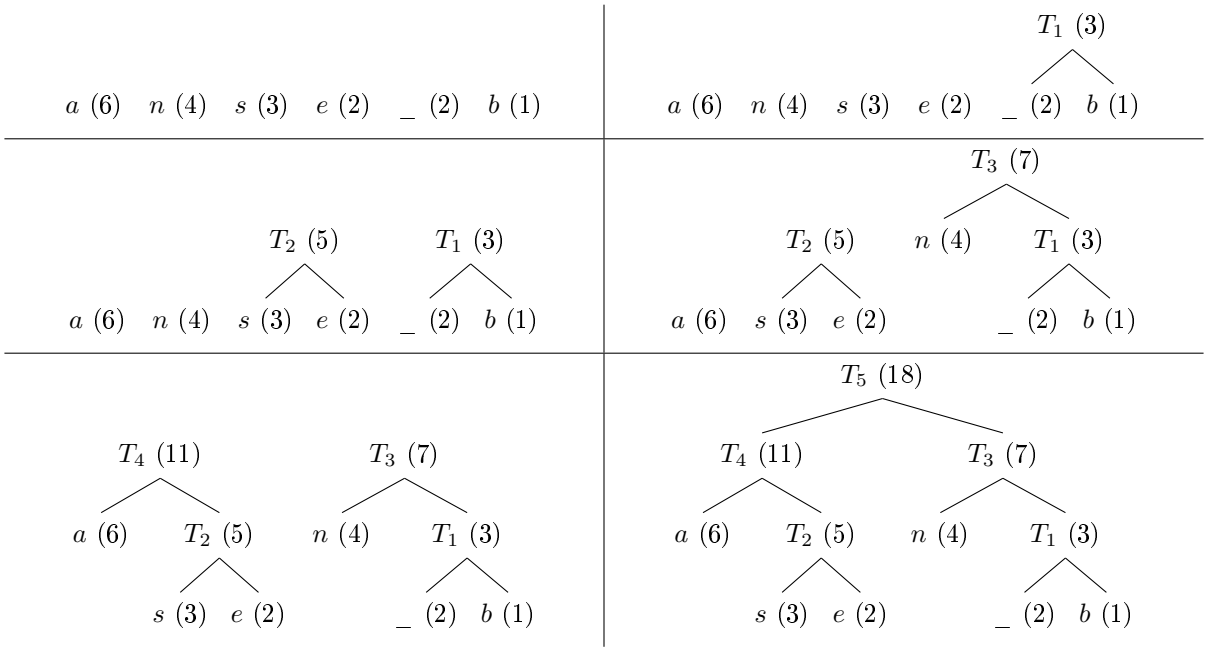
What is the minimum number of bits needed to represent the characters of the above phrase, considering the use of a constant-length coding system. Justify.

There are $N = 6$ unique characters, which are $\{a, b, e, n, s, \_\}$. To represent them with constant-length codes we need each code to have $\lceil \log_2 N \rceil \text{bit} = 3\,\text{bit}$.

Item b

Considering Huffman's compression algorithm with variable coding, present a binary coding for the characters in the phrase above and the tree you used to reach it.

Huffman coding produces a variable-length encoding so more frequent characters have smaller codes and less frequent characters have longer codes. It consists of finding the frequency of each character, and then successively join the two least frequent codes until all characters make up a binary tree.



Thus the encoding table is:

Character	Frequency	Code
a	6	00
n	4	10
s	3	010
e	2	011
_	2	110
b	1	111

Item c

What is the coding cost of the above phrase using constant coding? And what is its minimum cost, using variable coding? And what would it be the cost of using UTF-16 standard coding (16 bit per char)?

The coding cost of the phrase using constant coding is $18 * 3\,\text{bit} = 54\,\text{bit}$.

The coding cost of the phrase using variable coding is $6 * 2\,\text{bit} + 4 * 2\,\text{bit} + 3 * 3\,\text{bit} + 2 * 3\,\text{bit} + 2 * 3\,\text{bit} + 1 * 3\,\text{bit} = 44\,\text{bit}$.

The coding cost of the phrase using UTF-16 is $18 * 16\,\text{bit} = 288\,\text{bit}$.

Question 4

A digital marketing company is planning to launch a campaign targetting influent people of a social network. A campaign is only considered to be successful if it reaches all target personalities. However, due to budget restrictions, it is not possible to contact all personalities directly. The company intends to take advantage of the fact they know some personalities are friends with others, and that if the campaign reaches a personality it will also reach his/her immediate friends (but not friends of friends). Is it possible to implement an efficient algorithm to find the list of personalities that should be directly contacted such that it results in a successful campaign while staying on budget?

Item a

Formalize the problem and rewrite it as a decision problem.

Given a set of influencers V and their friendships E (where influencers u and v are friends iff $(u, v) \in E$), find the smallest subset S of influencers such that each influencer is in S or is a friend of someone in S.

Given a set of influencers and their relationships, is there a set S with size $|S| \leq k$ such that each influencer is in S or is a friend of someone in S?

Item b

Check if there is an efficient solution to this problem, explaining the steps of your solution.

This problem is exactly the same as the undirected vertex cover (VC) problem, where influencers V are the nodes V' in VC, friendships E are the edges E' in VC, and the subset of influencers S such that each influencer is in S or friend of someone in S is the solution S' of VC, which is the least subset of VC nodes such that all nodes are *covered* by S' (i.e., are either in S' or are adjacent to at least one node that is in S').

As we know the VC problem is NP-complete, and since our problem is equivalent to VC we can deduce it is also NP-complete. That means there is not an *efficient* solution to this problem.

We thus spared us from going through the usual steps of proving the problem is NP and then reducing the VC problem to our problem, since the equivalence is trivial.

Exam normal 2016/17

Question 1

In a library, you intend to organize a set of n books into shelves. The books' relative order has been predetermined by the librarian and can't be changed. A book i is characterized by its width L_i and height A_i. The length of each shelf is L_P. The shelves' heights are adjustable, where the height of each shelf is the height of the tallest book. The cost of a particular arrangement is the sum of the heights of all shelves. The intention is to determine the arrangement that minimizes cost.

Item a

Implement (in C++ or pseudocode) a solution for the problem, using a greedy algorithm. Discuss its optimality and mention its time complexity. Justify.

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    // Input
    int LP, n; cin >> LP >> n;
    vector<int> L(n), A(n);
    for(int i = 0; i < n; ++i) cin >> L[i] >> A[i];
    // Process with greedy
    list<int> shelves;
    int total_height = 0; int cur_shelf = 0;
    int cur_shelf_w = 0; int cur_shelf_h = 0;
    for(int i = 0; i < n; ++n){
        if(cur_shelf_w+L[i] > LP){
            total_height += cur_shelf_h; shelves.push_back(i);
            cur_shelf++;
            cur_shelf_w = 0; cur_shelf_h = 0;
        }
        cur_shelf_w += L[i]; cur_shelf_h = max(cur_shelf_h, A[i]);
    }
    total_height += cur_shelf_h;
    // Print solution
    cout << total_height << endl;
    cout << shelves.size() << endl;
    for(const int &s: shelves) cout << s << "␣"; cout << endl;
    return 0;
}
```

This algorithm is not optimal, and this can be proven by a simple example, to which we will find the solution given by the algorithm and then describe a better solution: consider $L_P = 2$, $L = \langle 1, 1, 1 \rangle$ and $A = \langle 1, 2, 2 \rangle$. The greedy algorithm would result in the arrangement $shelf = \langle 0, 0, 1 \rangle$ where both shelfs 0 and 1 have height 2, amounting to a total height of 4. However there is a solution $shelf = \langle 0, 1, 1 \rangle$ where shelf 0 has height 1 and shelf 1 has height 2, amounting to a total height of 3. This means there is a better solution that the greedy algorithm has not found, thus it is not optimal

Its time complexity is $O(n)$, since there is only one **for** loop (with no nested **for** loops) where all operations are elementary (thus $O(1)$).

Item b

To describe a solution, we need only two variables:

- How many books we have placed so far, i
- The free width of the current shelf, w

For each solution we save three pieces of information:

- The height of the current shelf, h
- The total height of the arrangement, excluding the current shelf, H
- The keys of the previous solution, (i, w)

The base case is $S_0^0 = (h = 0, H = 0)$.
For each solution S_i^w we can construct two more solutions:

- If the shelf still has space (if $w + L_i \leq L_P$), $S_{i+1}^{w+L_i}$ can be $(h' = \max\{h, A_i\}, H' = H)$.
- If we decide to put the book in the next shelf, $S_{i+1}^{L_i}$ can be $(h' = A_i, H' = H + h)$.

The final solution is

$$\min_{0 \leq w \leq L_P} \{S_n^w.H + S_n^w.h\}$$

and the corresponding arrangement can be extracted by traversing the previous solutions.

Its time complexity is $O(n \cdot L_P)$, since there are $n \cdot L_P$ possibly different states, and the processing of one state improves two other solutions in time $O(1)$.

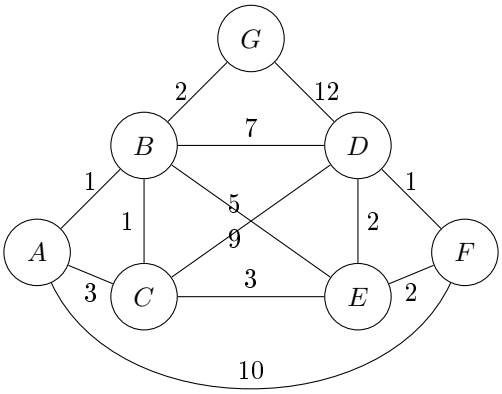
```cpp
#include <bits/stdc++.h>
using namespace std;

struct iw: public pair<int,int>{
    iw(int n1 = 0, int n2 = 0):pair<int,int>(n1,n2){}
    int i(){ return first; } int w(){ return second; }
};
struct sol_t { int h, H; iw prev;};
const sol_t null_solution = {-1, -1, iw(-1,-1)};

int main(){
    // Input
    int LP; cin >> LP;
    int n; cin >> n;
    vector<int> L(n), A(n);
    for(int i = 0; i < n; ++i) cin >> L[i] >> A[i];
    // Compute solution
    vector<vector<sol_t> > S(n+1,vector<sol_t>(LP+1, null_solution));
    S[0][0] = {0, 0, iw(-1,-1)};
    for(int i = 0; i < n; ++i){
        for(int w = 0; w < LP; ++w){
            const sol_t &s = S[i][w];
            if(s.h == -1) continue;
            if(w+L[i] < LP){
                sol_t &s1 = S[i+1][w+L[i]];
                if(s1.h == -1 || s.H < s1.H)
                    s1 = {max(s.h, A[i]), s.H, iw(i, w)};
            }
        }
    }
    // Get best solution
    iw best;
    int total_height = 1000000000;
    for(int w = 0; w <= LP; ++w){
        const sol_t &s = S[n][w];
        if(s.h + s.H < total_height){
            total_height = s.h + s.H; best = iw(n, w);
        }
    }
    // Track solutions to see where shelves start
    sol_t cur = S[best.i()][best.w()];
    list<int> shelves;
    for(int i = n; i > 0; --i){
        int H1 = S[cur.prev.i()][cur.prev.w()].H, H2 = cur.H;
        if(H1 != H2) shelves.push_back(i-1);
    }
    // Print solution
    cout << total_height << "\n";
    cout << shelves.size() << "\n";
    for(const int &s: shelves) cout << s << "␣"; cout << endl;
    return 0;
}
```

Question 2

Considering graph paths problems, answer the following items. When describing algorithms use pseudocode or C++.

Item a

Consider the undirected graph in the figure.



We want to determine the shortest paths, with origin A, generated by applying Dijkstra's algorithm. Show the values of distances of each vertex as they are processed. What is the shortest path from A to F?

Below is a trace of variables used in Dijkstra's algorithm.

Line	dist	Q	u
3	$\langle \infty, \infty, \infty, \infty, \infty, \infty, \infty \rangle$	$\{A, B, C, D, E, F, G\}$	-
4	$\langle 0, \infty, \infty, \infty, \infty, \infty, \infty \rangle$	$\{A, B, C, D, E, F, G\}$	-
5	$\langle 0, \infty, \infty, \infty, \infty, \infty, \infty \rangle$	$\{A, B, C, D, E, F, G\}$	-
8	$\langle 0, \infty, \infty, \infty, \infty, \infty, \infty \rangle$	$\{B, C, D, E, F, G\}$	A
5	$\langle 0, 1, 3, \infty, \infty, 10, \infty \rangle$	$\{B, C, D, E, F, G\}$	A
8	$\langle 0, 1, 3, \infty, \infty, 10, \infty \rangle$	$\{C, D, E, F, G\}$	B
5	$\langle 0, 1, 2, 8, 6, 10, 3 \rangle$	$\{C, D, E, F, G\}$	B
8	$\langle 0, 1, 2, 8, 6, 10, 3 \rangle$	$\{D, E, F, G\}$	C
5	$\langle 0, 1, 2, 8, 5, 10, 3 \rangle$	$\{D, E, F, G\}$	C
8	$\langle 0, 1, 2, 8, 5, 10, 3 \rangle$	$\{D, E, F\}$	G
5	$\langle 0, 1, 2, 8, 5, 10, 3 \rangle$	$\{D, E, F\}$	G
8	$\langle 0, 1, 2, 8, 5, 10, 3 \rangle$	$\{D, F\}$	E
5	$\langle 0, 1, 2, 7, 5, 7, 3 \rangle$	$\{D, F\}$	E
8	$\langle 0, 1, 2, 7, 5, 7, 3 \rangle$	$\{F\}$	D
5	$\langle 0, 1, 2, 7, 5, 7, 3 \rangle$	$\{F\}$	D
8	$\langle 0, 1, 2, 7, 5, 7, 3 \rangle$	$\{\}$	F
5	$\langle 0, 1, 2, 7, 5, 7, 3 \rangle$	$\{\}$	F
12	$\langle 0, 1, 2, 7, 5, 7, 3 \rangle$	$\{\}$	-

The shortest path from A to F is $A \to B \to C \to E \to F$, with a total cost of 7.

Item b

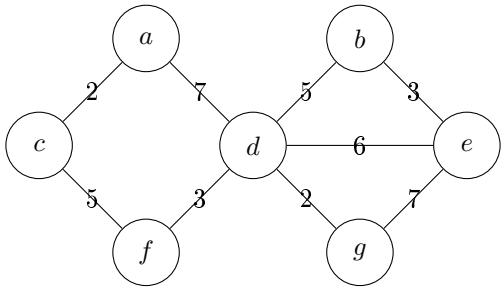
Present an efficient algorithm that, given a DAG (directed acyclic graph) with positive edges and vertices v_i, v_f abd v_k of that graph, finds the shortest path from v_i to v_f going through v_k. Suggestion: change Dijkstra's algorithm.

Algorithm 2 2017N-2b

```
1: function InDegrees(G(V, E))
2:     for v ∈ V do indegree(v) ← 0
3:     for e ∈ E do indegree(e.v) + +
4:     return indegree
5: function TopologicalSort(G)
6:     indegree ← InDegree(G)
7:     Q ← {v ∈ V | indegree(v) = 0}
8:     toporder ← []
9:     while |Q| > 0 do
10:        u ← Q.First()
11:        Q.Erase(u)
12:        toporder.Push(u)
13:        for v ∈ Adj(G, u) do
14:            − − indegree(v)
15:            if indegree(v) = 0 then Q ← Q ∪ {v}
16: Q ← []
17: function ShortestPath(G(V, E), s, d)
18:     for v ∈ V do
19:         dist(v) ← ∞
20:         prev(v) ← NULL
21:     dist(s) ← 0
22:     i ← Q.Find(s)
23:     while i < |Q| do
24:         u ← Q_i
25:         + + i
26:         if u = d then break
27:         for v ∈ Adj(G, u) do
28:             c′ ← dist(u) + w(u, v)
29:             if c′ < dist(v) then
30:                 dist(v) ← c′, prev(v) ← u
31:     if prev(d) = NULL then return []
32:     ret ← []
33:     while d ≠ s do
34:         ret.Push(d)
35:         d ← prev(d)
36:     return dist, prev
37: function ShortestPathMid(G(V, E), v_i, v_f, v_k)
38:     Q ← TopologicalSort(G)
39:     return ShortestPath(G, v_i, v_k) + ShortestPath(G, v_k, v_f)[1 : end]
```

Question 3

Consider the weighted graph below and answer the following items.



Item a

Mention all articulation points of the graph, if there is any. Explain.

An articulation point, also known as a cut vertex, is a vertex that, if removed, would disconnect a strongly connected graph.

This graph is strongly connected, so there is a chance there are articulation points.

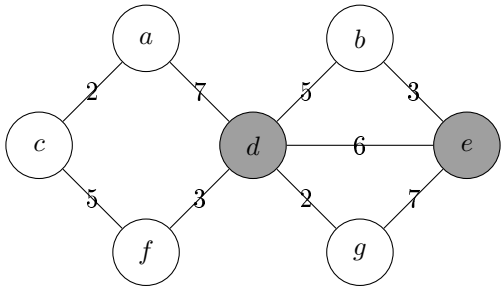
If d is removed, the graph will become disconnected, having as strongly connected components $\{a, c, f\}$ and $\{b, e, g\}$.

This is the only articulation point, since there is no other vertex that would disconnect the graph when removed.

Item b

Find an "optimal Chinese postman path", starting in vertex a. Explain, step by step, the method you used to calculate that path and why it is optimal.

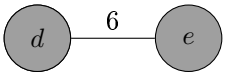
Step 1: find all vertices of odd degree.



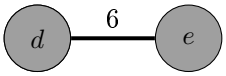
Step 2: Find shortest paths between all pairs of odd-degree vertices.

$d(u,v)$	d	e
d	-	6
e	-	-

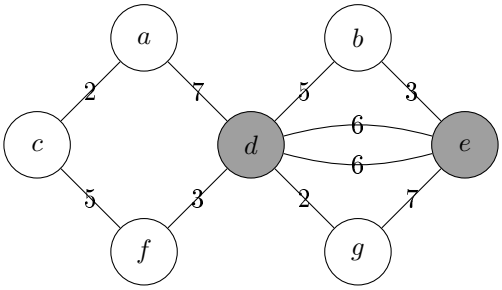
Step 3: Build a complete graph G' with odd-degree vertices and edges equal to the minimum distances calculated in 2.



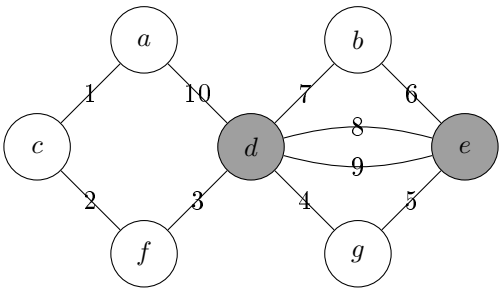
Step 4: Find a minimum-weight perfect pairing.



Step 5: For each pair (u, v) in the perfect pairing, add pseudo-edges to G and call it G^*



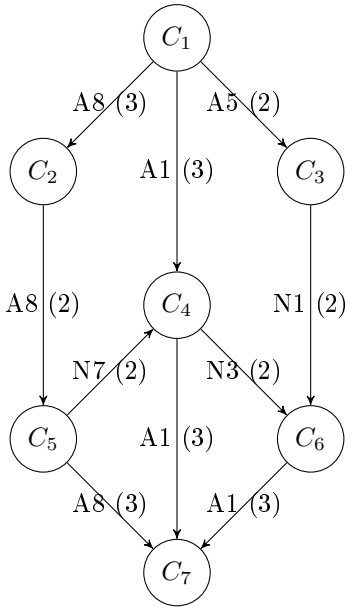
Step 6: Find an Euler circuit in G^*.



$a \to c \to f \to d \to g \to e \to b \to d \to e \to d \to a$

Question 4

A country's government decided to build a highway network connecting cities C_1, C_2, C_3, C_4, C_5, C_6 and C_7, as shown in the graph.

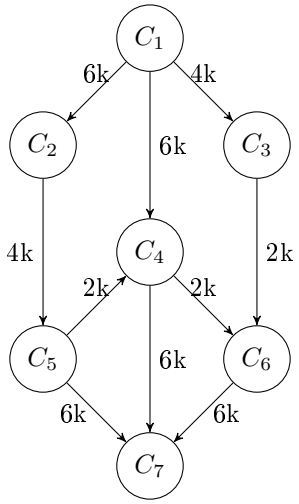


Some parts of the network were classified as *highways* (A) and others as *national roads* (N). In parenthesis are the number of traffic lanes, which limits the maximum number of vehicles that can traverse it — in the case of highways, each lane admits an hourly volume of 2,000 vehicles, while national road lanes admit only 1,000 vehicles. It has been estimated that the number of hourly travels between C_1 and C_7 is 18,000 vehicles per hour.

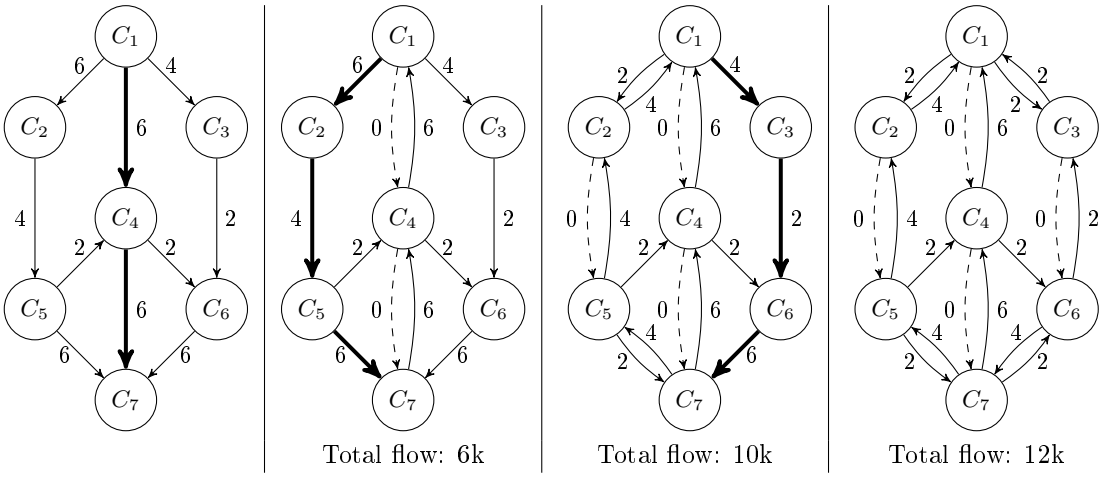
Item a

Check if the network can sustain the hourly number of estimated travels. Justify.

This is the actual hourly vehicle maximum flow through each road:



To reach an answer, we will apply the Edmonds-Karp algorithm, which consists of selecting the shortest augmenting path until there are no augmenting paths.



As we can see, the maximum hourly vehicle flow from C_1 to C_7 is 12,000 vehicles, so the network cannot sustain the hourly number of estimated travels of 18,000 vehicles per hour.

Item b

If the govenment allowed a company to explore the network by charging road tolls in one of the roads, which would be the most profitable? Justify.

Diogo Miguel Ferreira Rodrigues (dmfrodrigues2000@gmail.com)

The expected behaviour when a toll is installed is for travellers to avoid that road as much as possible, so we are looking for the road that, when tolled, gives the most profit (i.e., has the most hourly vehicle flow) taking into account travellers will try to avoid it at all cost.

By the previous item, we can visualize this graph as a bipartite graph, where the source subgraph is $\{C_1, C_2, C_3\}$, the sink subgraph is $\{C_4, C_5, C_6, C_7\}$ and the edges that cross from source to sink subgraphs are $C_2 \to C_5$ (4k), $C_1 \to C_4$ (6k) and $C_3 \to C_6$ (2k). Since these three edges are thus unavoidable if the flow is 12k (the maximum the network can sustain), we will be looking forward to toll one of these roads.

Now we just need to choose the one with the most hourly car flow, so the final answer is $C_1 \to C_4$, with a capacity of 6k vehicles per hour.

Question 5

The genetic code uses 4 letters (A, C, G, T) to code the bases that make up DNA. Gene XPTO is made of the following base sequence, and respective letter frequency:

A A G G T A C C T A C C C C C C C C C C A (5 A's, 13 C's, 2 G's, 2 T's)

Item a

Considering a coding system of fixed size, what is the smallest code, enough to represent the alphabet? What is the size, in bits, to code gene XPTO? Explain.

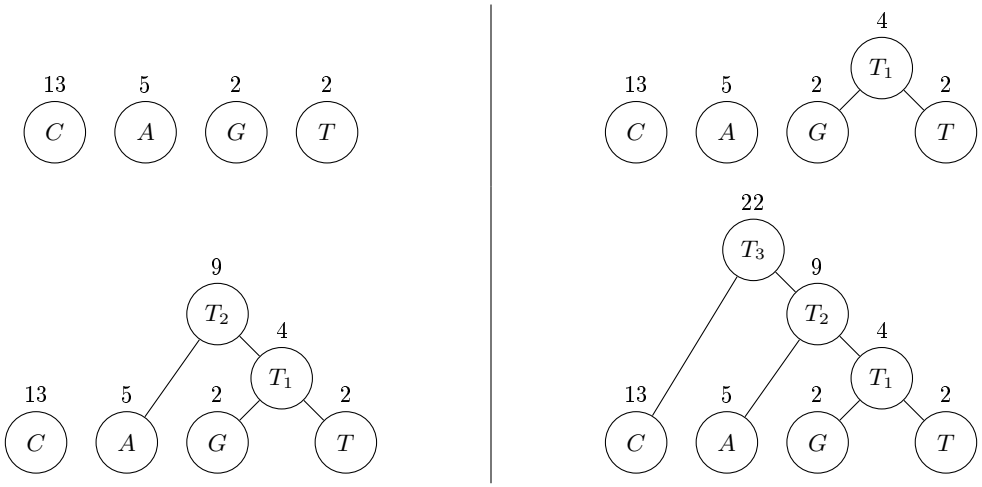
Since the alphabet has $N = 4$ letters we need $\lceil \log_2 N \rceil \text{bit} = \lceil 2 \rceil \text{bit} = 2\,\text{bit}$ bits to represent each letter in the alphabet. To encode gene XPTO, we would need $2\,\text{bit} * 22 = 44\,\text{bit}$.

Item b

Considering variable-length codes, what is the least total cost of coding gene XPTO? Explain in detail.

To find the least total cost of coding gene XPTO, we will use Huffman encoding, as it theoretically provides the least cost coding. This algorithm assigns longer codes to less frequent characters, and shorter codes to more frequent characters.

In this case, we already have the frequencies, so we can immediately build the binary tree, by successively joining less-frequent characters:



This means the encoding table is

Character	Code
C	0
A	10
G	110
T	111

Thus, gene XPTO is encoded to

10 10 110 110 111 10 0 0 111 10 0 0 0 0 0 0 0 0 0 0 10

which has a length of $5 * 2\,\text{bit} + 13 * 1\,\text{bit} + 2 * 3\,\text{bit} + 2 * 3\,\text{bit} = 35\,\text{bit}$, which is an improvement compared to 44 bit by constant-length codes.

Question 6

In an informatics summer camp, several interesting courses are offered to participants, and are taught by internationally recognized lecturers from well known universities and companies. Participants can sign up for several of those courses, without limitation as to the number of courses they choose. In the meanwhile, participants will only have course certificates for those they are approved in the final exam. Considering all final exams last one hour, and are only made once, students from different courses cannot have overlapping exams. Is it possible to implement an efficient algorithm to determine the least number of 1-hour slots so as to avoid students signed up in several courses to have overlapping exams?

Considering this problem, answer the following items:

Item a

Rewrite this problem as a decision problem.

Given a set of students and a set of exams for each student, as well as a number N, is there an acceptable exam schedule (no student has overlapping exams) that takes less than or N 1-hour slots?

Item b

Check if there is an efficient solution to this problem, explaining the steps of your solution.

This problem is NP-complete since:

- It is NP, as a solution for the decision problem can be checked in polynomial time; one has to (1) check the number of slots is less or equal to N, and (2) for each student assert he/she does not have exams taking place in the same slot.
- It is NP-hard, as the graph colouring decision problem (k colours) is reducible to this problem in polynomial time (if there is a solution to this problem then there is also a solution for graph colouring; thus this problem is at least as hard as the graph colouring problem):
  - **Input conversion:** a node in graph colouring is an exam; an edge in graph colouring is a student; the maximum k colours in graph colouring are the maximum k 1-hour slots we can use in our decision problem's witness.
  - **Output conversion:** a 1-hour slot is a color in graph colouring.
  - Similarly to the graph colouring restriction that an edge cannot start and end in vertices of the same color, a student cannot have two exams in the same 1-hour slot.

Exam resource 2016/17

Question 1

Consider a sequence of natural numbers S, for instance $S = \langle 8, 2, 6, 7, 1, 2, 3, 4, 5, 4, 5, 3, 1 \rangle$. We want to find the minimum number of subsequences (of contiguous elements) such that each subsequence starts and ends with the same number.

In the presented example, the minimum number is 4, and the subsequences are ($\langle 8 \rangle$, $\langle 2, 6, 7, 1, 2 \rangle$, $\langle 3, 4, 5, 4, 5, 3 \rangle$ and $\langle 1 \rangle$).

Item a

Implement (in C++ or pseudocode) a solution for this problem, using a greedy algorithm and explain. Mention its time complexity. Justify.

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    // Input
    size_t N; cin >> N;
    vector<int> S(N);
    for(size_t i = 0; i < N; ++i)
        cin >> S[i];
    // Get last occurence of each digit
    unordered_map<int, size_t> last_occurence;
    for(ssize_t i = N-1; i >= 0; --i)
        if(!last_occurence.count(S[i]))
            last_occurence[S[i]] = i;
    // Create sequences
    list<size_t> new_sequence;
    size_t l = 0;
    while(l < N){
        l = last_occurence[S[l]]+1;
        new_sequence.push_back(l);
    }
    new_sequence.pop_back();
    // Print
    cout << new_sequence.size() << endl;
    for(const size_t &i: new_sequence)
        cout << i << endl;
    return 0;
}
```

This algorithm first finds the last occurence of every number in the sequence, and then processes the sequence again and expands each sequence individually to as close to the end as possible (so given we have already processed the first i numbers and we are processing number S_i, we will greedily extend this sequence until that number's last occurence).

Calculating the last occurence of each number in the sequence takes time $O(N)$ amortized, since we iterate over S in reverse order and `unordered_map` operations are time $O(1)$ amortized. The final processing of S takes at most $O(N)$ for an array where all numbers appear only once.

Item b

Formalize a solution for this problem, using dynamic programming. Implement (in C++ or pseudocode) that solution and explain. Mention its time complexity. Justify.

To uniquely identify a state we need to know:

- i, the position of the number we are about to process.
- s, the first number of the sequence we are currently building.

A state's solution T_i^s is the number of sequences so far.
From a state T_i^s we can generate two more solutions:

- $T_{i+1}^s = T_i^s$, by simply considering i is in the current subsequence.
- $T_{i+1}^{S_i} = T_i^s + 1$ if $s = S_{i-1}$, so we end the current subsequence and create a new one starting in i.

The base case is $T_1^{S_0} = 0$, which means we assigned the first element to a sequence and have 0 complete sequences.

The final solution is $T_N^{S_{N-1}} + 1$, which translates to "having analysed all N elements of S, what is the least number of sequences such that the first element of the current (and last) sequence is equal to the last element" (this is so that we can end the last sequence in the last element).

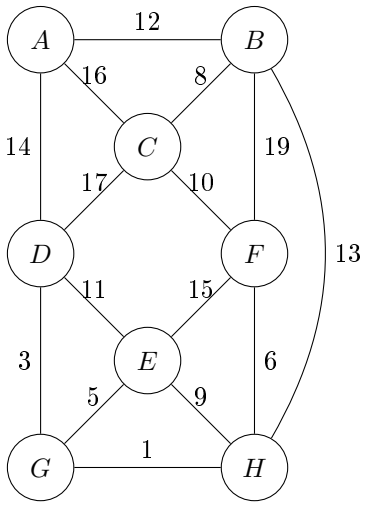
```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  int main(){
5      // Input
6      size_t N; cin >> N;
7      vector<int> S(N); for(size_t i = 0; i < N; ++i) cin >> S[i];
8      // Process
9      vector< unordered_map<int, size_t> > T(N+1);
10     vector< unordered_map<int, int> > prev(N+1);
11     T[1][S[0]] = 0; prev[0][0] = -1;
12     for(size_t i = 1; i < N; ++i){
13         for(const auto &s_: T[i]){
14             const int &s = s_.first;
15             if(T[i+1].count(s) == 0 || T[i+1][s] > T[i][s]){
16                 T[i+1][s] = T[i][s]; prev[i+1][s] = s;
17             }
18             if(s == S[i-1] &&
19               (T[i+1].count(S[i]) == 0 || T[i+1][S[i]] > T[i][s] + 1)){
20                 T[i+1][S[i]] = T[i][s] + 1; prev[i+1][S[i]] = s;
21             }
22         }
23     }
24
```

```
25      // Get output
26      list<size_t> new_sequence;
27      int current_s = S[N-1];
28      for(ssize_t i = N; i > 1; --i){
29          int prev_s = prev[i][current_s];
30          if(T[i-1][prev_s] != T[i][current_s]){
31              new_sequence.push_front(i-1);
32          }
33          current_s = prev_s;
34      }
35      // Print
36      cout << new_sequence.size() << endl;
37      for(const size_t &i: new_sequence) cout << i << endl;
38      return 0;
39  }
```

Its time complexity is $O(|S| \cdot M)$ amortized where M is the maximum value of numbers in S. This is because our states are identified by $1 \leq i \leq |S|$ and $1 \leq s \leq M$ so we have at most exactly $|S| \cdot M$ states, and each state can produce two more states in constant time. Also, access to the second dimension of T is made via an `unordered_map`, so its access time is $O(1)$ amortized.

Question 2

Consider the following weighted undirected graph. Edge weights are integers w where $0 < w < 20$.

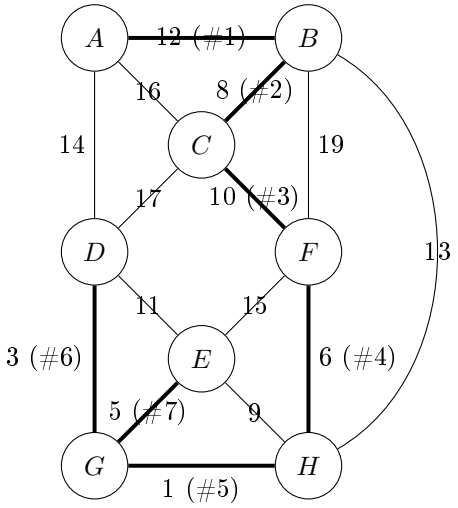


Item a

Get the sequence of edges of the minimum spanning tree, in the order by which they are considered when Prim's algorithm is applied. Justify. Consider A to be the starting vertex.

Below is a simplified trace of Prim's algorithm in this graph.

Visited	**Adjacent edges**	**Chosen edge**	**Weight**
A	$A \xrightarrow{12} B$, $A \xrightarrow{16} C$, $A \xrightarrow{14} D$	$A \xrightarrow{12} B$	12
A, B	$A \xrightarrow{16} C$, $A \xrightarrow{14} D$, $B \xrightarrow{8} C$, $B \xrightarrow{19} F$, $B \xrightarrow{13} H$	$B \xrightarrow{8} C$	20
A, B, C	$A \xrightarrow{14} D$, $B \xrightarrow{19} F$, $B \xrightarrow{13} H$, $C \xrightarrow{17} D$, $C \xrightarrow{10} F$	$C \xrightarrow{10} F$	30
A, B, C, F	$A \xrightarrow{14} D$, $B \xrightarrow{13} H$, $C \xrightarrow{17} D$, $F \xrightarrow{15} E$, $F \xrightarrow{6} H$	$F \xrightarrow{6} H$	36
A, B, C, F, H	$A \xrightarrow{14} D$, $C \xrightarrow{17} D$, $F \xrightarrow{15} E$, $H \xrightarrow{9} E$, $H \xrightarrow{1} G$	$H \xrightarrow{1} G$	37
A, B, C, F, G, H	$A \xrightarrow{14} D$, $C \xrightarrow{17} D$, $F \xrightarrow{15} E$, $H \xrightarrow{9} E$, $G \xrightarrow{3} D$, $G \xrightarrow{5} E$	$G \xrightarrow{3} D$	40
A, B, C, D, F, G, H	$D \xrightarrow{11} E$, $F \xrightarrow{15} E$, $H \xrightarrow{9} E$, $G \xrightarrow{5} E$	$G \xrightarrow{5} E$	45
A, B, C, D, F, G, H			



The MST edges, by the order they were added to the MST, is A—B, B—C, C—F, F—H, H—G, G—D and G—E.

Item b

A new edge C—E is added to the graph. What is the possible range of values for its weight so that C—E belongs to the MST? Explain.

To guarantee (C, E) belongs to the MST found by Prim's algorithm, it would have to have a weight of 9 or less. This is because edge #3 has weight 10 and all following edges have weight less than 10, and to guarantee an edge (u, v) is added one must guarantee it will be the edge with least weight in one of the possible edge choices after either u or v have been added to the MST. Therefore, our best chances are to have $w(C, E) < 10$ so that it is selected as edge #3.

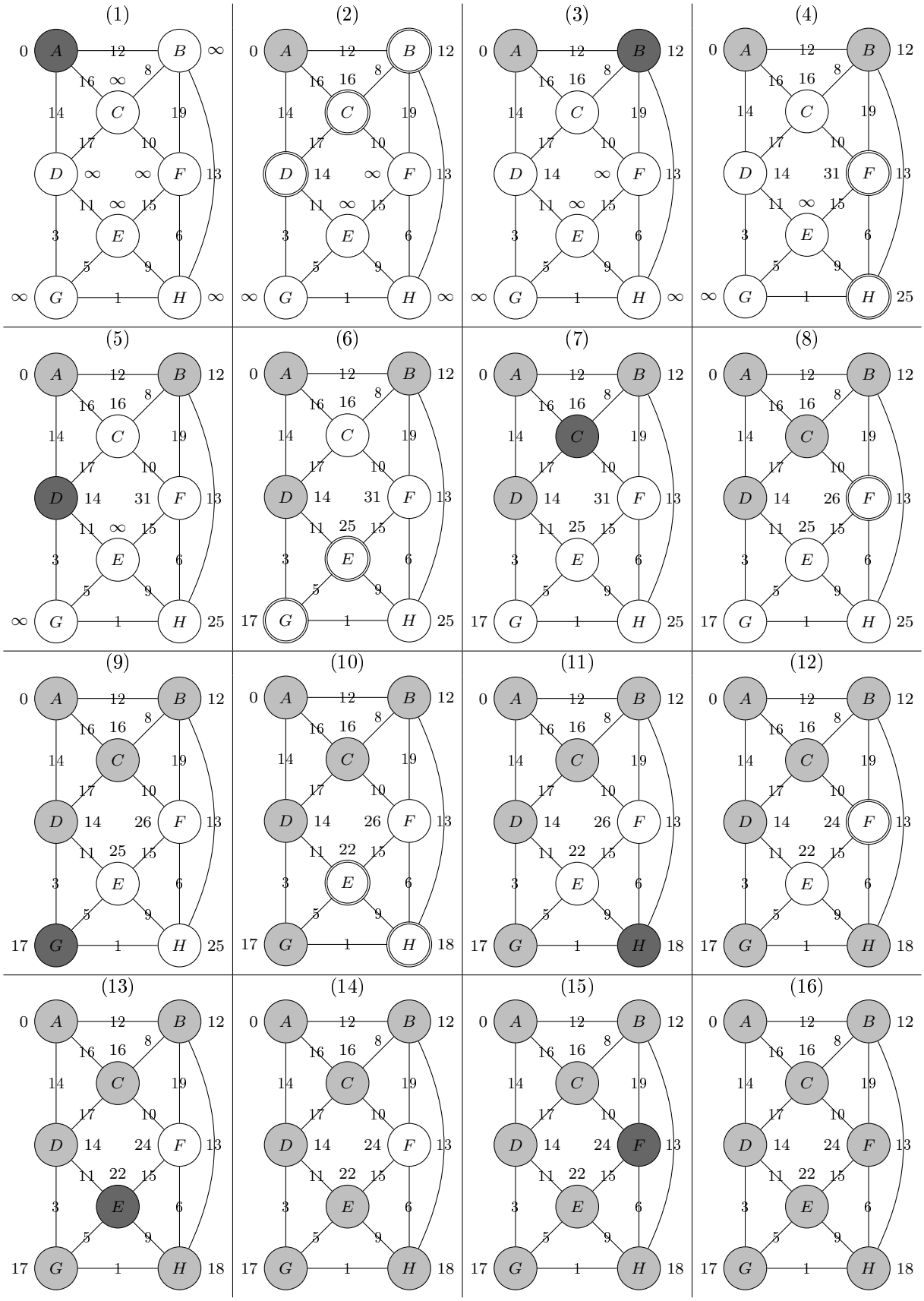
Question 3

Consider the previous question's graph, representing a map, where vertices identify points of historic/-landscape interest and edges present distances between points.

Item a

Mr. Santos is on vacation in the hotel at A and wants to visit some points of interest during the days he will be staying there, so he is interested on knowing the shortest paths originating from A. Applying Dijkstra's algorithm, present distances to each vertex in the graph as they are processed. What is the shortest path from A to H?

The shortest path from A to H is $A \to D \to G \to H$, with a total weight of 18.


(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
(9)
(10)
(11)
(12)
(13)
(14)
(15)
(16)

Item b

Mr. Santos now wants to go for a walk to point H, visiting the minimum number of different points, since passing through a point requires paying a tariff. Describe in pseudocode an efficient algorithm to help Mr. Santos plan his walk. What is the time complexity of the algorithm? Justify.

This problem is equivalent to the shortest path problem in an undirected unweighted graph, so we can use a breadth-first search to find the shortest path.

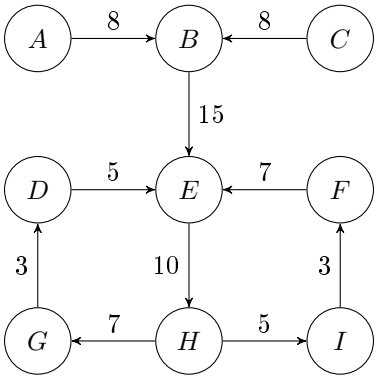
Algorithm 3 2017R-3b

```
1:  function BFS(G(V, E), s)
2:      for v ∈ V do dist(v) ← ∞, prev(v) ← NULL
3:      Q ← Queue()
4:      dist(s) ← 0, Q.Push(s)
5:      while |Q| > 0 do
6:          u ← Q.Front()
7:          Q.Pop()
8:          if u = s then break
9:          for v ∈ Adj(G, u) do
10:             c' ← dist(u) + 1
11:             if c' < dist(v) then
12:                 dist(v) ← c', prev(v) ← u
13:     return dist, prev
```

This algorithm has complexity $O(|V| + |E|)$, because it processes every node exactly once, and every edge exactly twice (edge (u, v) is processed once when processing u and again when processing v).

Question 4

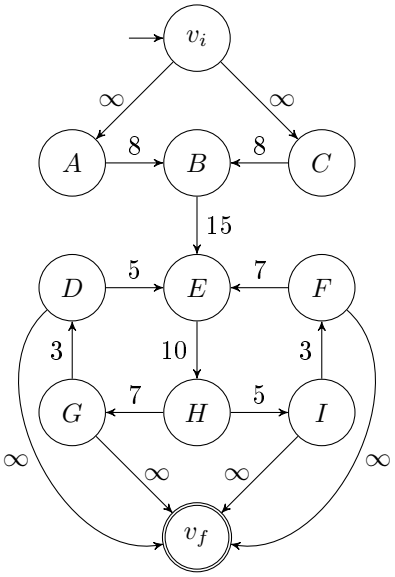
An irrigation system was projected with the following configuration, where pipe capacities are indicated by the integer values next to the respective edges. There are two pumps feeding the irrigation system at a constant rate (litres/hour), placed at vertices A and C. Sprinklers have been installed at vertices D, F, G and I. Answer the following items, adequately justifying your answer.



Item a

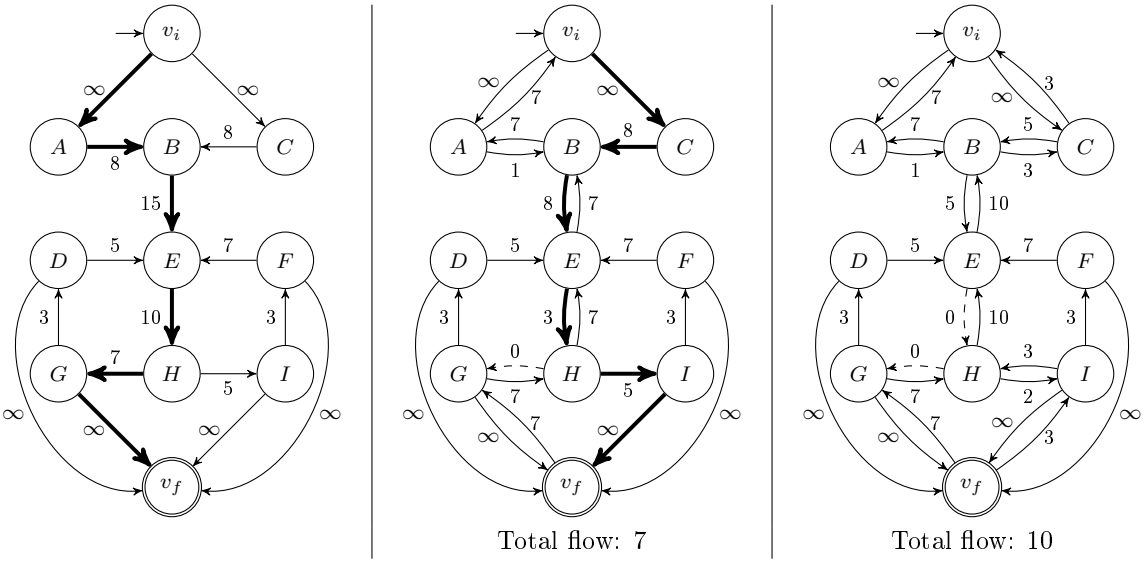
What is the maximum flow feeding the whole irrigation system?

By creating a single source and sink for this system we can have a better overview of it.



We will use the Edmonds-Karp algorithm to find the maximum flow. This algorithm consists of selecting the shortest augmenting path and then proceed as usual for Ford-Fulkerson, decrementing residues and increasing flows.

Below is a trace, showing the residues graph.



Item b

What will be the flow through $E \to H$ if a sprinkler is instaled in vertex E?

If a sprinkler is installed in E, the only two augmenting paths the Edmonds-Karp algorithm will select are $v_i \to A \to B \to E \to v_f$ with a flow of 8, and $v_i \to C \to B \to E \to v_f$ with a flow of 7. After that, all flow from $B \to E$ will be exausted, so there is no more flow. That means all other pipes, including $E \to H$, will have 0 flow.

Diogo Miguel Ferreira Rodrigues (dmfrodrigues2000@gmail.com)

Question 5

Consider the evolutional distance between two genes is the minimum number of mutations that transform a gene into another. According to this concept, answer the following questions, justifying your answers.

Item a

What is the evolutional distance between the two following genes:

- XPTO: A G G T A C T A C C C C A
- OPTX: A A G G A C A C C C C A

Consider a mutation can either be an insertion, a deletion or the replacement of a DNA base.

We will use a dynamic programming approach to solve this problem. Each state is uniquely identified by:

- i, length of the first string being considered.
- j, length of the second string being considered.

D_i^j is the edit distance between $P[0:i)$ and $T[0:j)$, so the recursive formula is:

$$D_i^j = \min \begin{cases} D_{i-1}^{j-1} & : P_i = T_j \\ D_{i-1}^{j-1} + 1 & (\text{replace } T_j \text{ by } P_i) \\ D_{i-1}^{j} + 1 & (\text{insert } P_i \text{ after } T_j) \\ D_{i}^{j-1} + 1 & (\text{erase } T_j) \end{cases}$$

The base cases are $D_0^j = j$ and $D_i^0 = i$, which are the costs of creating one of the strings from nothing. The solution is D_M^N where $|P| = M$ and $|T| = N$.

	T		A	A	G	G	A	C	A	C	C	C	C	A
P		0	1	2	3	4	5	6	7	8	9	10	11	12
	0	**0**	**1**	2	3	4	5	6						
A	1	1	0	**1**	2	3	4	5						
G	2	2	1	1	**1**	2	3	4						
G	3	3	2	2	1	**1**	2	3						
T	4	4	3	3	2	2	**2**	3						
A	5	5	4	4	3	3	**2**	3						
C	6	6	5	5	4	4	3	**2**						
T	7	7	6	6	5	5	4	**3**						
A	8								**3**					
C	9									**3**				
C	10										**3**			
C	11											**3**		
C	12												**3**	
A	13													**3**

The evolutional distance between the two genes is 3.

Item b

Define the prefix-function to use the Knuth-Morris-Pratt algorithm to search gene G T G C C in a DNA sequence.

$$\pi_q = \max\{k : k < q \wedge P[0:k) = P[q-k:q)\}$$

q	1	2	3	4	5
P_q	G	T	G	C	C
π_q	0	0	1	0	0

Question 6

In graph theory, a *clique* of an undirected graph is a subset of its vertices such that, for each pair of vertices (u, v), there is an edge connecting them. Formally, given an undirected graph $G = (V, E)$, a subset $V_C \subseteq V$ is a *clique* of graph G iff $\forall u, v \in V_C \implies (u, v) \in E$.

Considering this problem, answer the following questions:

Item a

Rewrite the clique problem C in undirected graphs as a decision problem.

Given an undirected graph $G = (V, E)$, does it have a clique V_C (V_C is a clique iff $V_C^2 \subseteq E$) with size $|V_C|$ larger than a given k?

Item b

Check if there is an efficient solution for this problem, briefly explaining your process' steps.

Suggestion: If needed, you can use the following definitions of NP-complete problems, as well as consider other NP-complete problems you may know.

Graph colouring (GC): Given a graph $G = (V, E)$, to colour graph G is to assign labels (or colors) to all vertices of G so that there is no edge $(i, j) \in E$ for which i and j have the same label (or color); that is, $color(i) \neq color(j)$.

Independent set (IS): Given a graph $G = (V, E)$, an independent set of vertices of G is a subset $V_i \subseteq V$ such that there are not two vertices $i, j \in V_i$ where $(i, j) \in E$.

There is not an efficient solution to this problem, given it is NP-complete. This is because:

- it is in NP, since we can check if a set is a clique in polynomial time, by iterating over all pairs (i, j) and check if they exist in E.
- the IS problem is reducible to the clique problem, if we consider that:
  - A graph $G(V, E)$ in the IS problem is a graph $G'(V, E')$ in the clique problem, where $E' = \overline{E} = V^2 \backslash E$ (E' is the complement of E).
  - A clique V_C in the clique problem is an independent set V_i in the IS problem, because an independent set in $G(V, E)$ is $V_i \subseteq V : \forall i, j \in V_i, (i, j) \notin E$ and a clique in $G'(V, E')$ is $V_C \subseteq V : \forall i, j \in V_i, (i, j) \in E'$, and because $E' = \overline{E}$ we trivially know that $(i, j) \notin E \iff (i, j) \in E'$.

Exam normal 2015/16

Question 1

Assume you are managing the process of placing outdoor billboards in a part of the A1 highway (north-south direction) with N kilometers. Vector `pos[]` specifies possible places for billboards, where `pos[i]` is the distance in km from point i to the end of the road (assume this vector to be sorted in increasing order). On placing a billboard, a value is charged which depends on the place the billboard is placed. Consider `value[]` to be the vector representing values to be charged, where `value[i]` is the value to be charged for placing a billboard at position `pos[i]`. You cannot place billboards less than 5 km appart. You want to find the maximum value that can be charged for billboards.

Item a

Implement (in C++ or pseudocode) a solution for this problem, using a greedy algorithm. Mention its time complexity. Is that algorithm optimal? Explain.

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    // Input
    int N; cin >> N;
    vector<int> pos(N), value(N);
    for(int i = 0; i < N; ++i) cin >> pos[i] >> value[i];
    // Processing
    list<int> billboards;
    int max_value = 0;
    billboards.push_back(0); max_value += value[0];
    for(int i = 1; i < N; ++i)
        if(pos[i] - pos[*billboards.rbegin()] >= 5){
            billboards.push_back(i); max_value += value[i];
        }
    // Output
    cout << max_value << endl;
    cout << billboards.size() << endl;
    for(const int &i: billboards) cout << i << "␣";
    cout << endl;
    return 0;
}
```

This algorithm runs in time $O(N)$, since it iterates once over each of the N potential billboard places. This algorithm works by selecting the billboard with smallest `pos` that complies with the 5 km rule.

This algorithm is not optimal. To prove that, we will present a test case for which our algorithm produces a sub-optimal solution: $pos = \langle 1, 2 \rangle$, $value = \langle 1, 10 \rangle$. Our algorithm would only choose to place a billboard at 0 (with profit 1), but the solution of placing a billboard at 1 (with profit 10) is better, thus our algorithm has not found the optimal solution, otherwise it would be impossible to mention a better solution.

Item b

Formalize a solution to this problem using dynamic programming. Implement (in C++ or pseudocode) that solution and mention its time complexity. Explain.

Suggestion: if you wish, you may consider a value `e[j]` as being the closest place from `pos[j]` whose distance to `j` is greater than or equal to 5 km.

A state is uniquely represented by:

- i, the place we are about to process.
- d, the location from which we can install the next billboard.

S_i^d is the profit we make in state (i, d).

The base case is $S_0^0 = 0$, meaning we have not yet installed any billboards and are deciding to install a billboard or not in the first place.

Each state (i, d) can contribute to two more states:

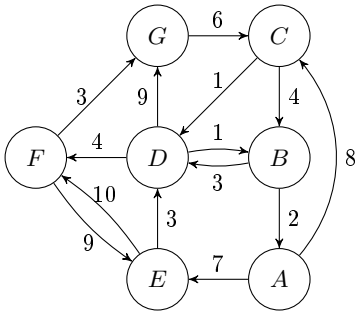
- If a billboard can be placed ($d \leq pos[i]$) then $S_{i+1}^{pos[i]+5-1}$ can be $S_i^d + value[i]$.
- $S_{i+1}^d = S_i^d$ if we do not place a billboard at i.

The solution is $\max_{d \in \mathbb{R}} \{S_N^d\}$.

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    // Input
    int N; cin >> N;
    vector<int> pos(N), value(N);
    for(int i = 0; i < N; ++i) cin >> pos[i] >> value[i];
    vector<int> e(N); int current_i = 0;
    for(int i = 0; i < N; ++i){
        while(current_i < N && pos[current_i] - pos[i] < 5) ++current_i;
        e[i] = current_i;
    }
    // Processing
    vector< unordered_map<int, int> > S(N+1), prev(N+1);
    S[0][0] = 0;
    for(int i = 0; i <= N; ++i)
        for(const auto &p: S[i]){
            const int &d = p.first;
            int d_ = pos[i]+5-1;
            if(d <= pos[i] && S[i+1][d_] < S[i][d] + value[i]){
                S[i+1][d_] = S[i][d] + value[i]; prev[i+1][d_] = d;
            }
            if(S[i+1][d] < S[i][d]){
                S[i+1][d] = S[i][d]; prev[i+1][d] = d;
            }
        }
    // Get best answer
    int current_d = 0;
    int max_value = -1;
    for(const auto &p: S[N]){
        const int &d = p.first;
        if(S[N][d] > max_value){
            max_value = S[N][d]; current_d = d;
        }
    }
    // Backtrack
    list<int> billboards;
    for(int i = N; i > 0; --i){
        int prev_d = prev[i][current_d];
        if(S[i][current_d] != S[i-1][prev_d]){
            billboards.push_front(i-1);
        }
        current_d = prev_d;
    }
    // Output
    cout << max_value << endl;
    cout << billboards.size() << endl;
    for(const int &i: billboards) cout << i << "␣";
    cout << endl;
    return 0;
}
```

Question 2

The following directed graph represents a map, where vertices are cities and the value of an edge is the distance between adjacent cities. Mr Joaquim lives in city A and intends to visit his family in city G.



Item a

Find the path that Mr Joaquin should follow to travel the least distance. Explain.

We will find the shortest path from A to G using Dijkstra's algorithm. The following table is a trace for Dijkstra's algorithm.

Algorithm 4 Dijkstra's algorithm

```
1:  function DIJKSTRA(G(V, E), s)
2:      for v ∈ V do dist(v) ← ∞, prev(v) ← NULL
3:      Q ← V
4:      dist(s) ← 0
5:      while |Q| > 0 do
6:          u ← vertex of Q with least dist(u)
7:          Q ← Q\{u}
8:          for v ∈ ADJ(G, u) do
9:              c' ← dist(u) + w(u, v)
10:             if c' < dist(v) then
11:                 dist(v) ← c', prev(v) ← u
12:     return dist, prev
```

Line	$(dist, prev)$	Q	u
3	$\langle(\infty,-),(\infty,-),(\infty,-),(\infty,-),(\infty,-),(\infty,-),(\infty,-)\rangle$	$\{A,B,C,D,E,F,G\}$	-
5	$\langle(0,-),(\infty,-),(\infty,-),(\infty,-),(\infty,-),(\infty,-),(\infty,-)\rangle$	$\{A,B,C,D,E,F,G\}$	-
8	$\langle(0,-),(\infty,-),(\infty,-),(\infty,-),(\infty,-),(\infty,-),(\infty,-)\rangle$	$\{B,C,D,E,F,G\}$	A
5	$\langle(0,-),(\infty,-),(8,A),(\infty,-),(7,A),(\infty,-),(\infty,-)\rangle$	$\{B,C,D,E,F,G\}$	A
8	$\langle(0,-),(\infty,-),(8,A),(\infty,-),(7,A),(\infty,-),(\infty,-)\rangle$	$\{B,C,D,F,G\}$	E
5	$\langle(0,-),(\infty,-),(8,A),(10,E),(7,A),(17,E),(\infty,-)\rangle$	$\{B,C,D,F,G\}$	E
8	$\langle(0,-),(\infty,-),(8,A),(10,E),(7,A),(17,E),(\infty,-)\rangle$	$\{B,D,F,G\}$	C
5	$\langle(0,-),(12,C),(8,A),(9,C),(7,A),(17,E),(\infty,-)\rangle$	$\{B,D,F,G\}$	C
8	$\langle(0,-),(12,C),(8,A),(9,C),(7,A),(17,E),(\infty,-)\rangle$	$\{B,F,G\}$	D
5	$\langle(0,-),(10,D),(8,A),(9,C),(7,A),(13,D),(18,D)\rangle$	$\{B,F,G\}$	D
8	$\langle(0,-),(10,D),(8,A),(9,C),(7,A),(13,D),(18,D)\rangle$	$\{F,G\}$	B
5	$\langle(0,-),(10,D),(8,A),(9,C),(7,A),(13,D),(18,D)\rangle$	$\{F,G\}$	B
8	$\langle(0,-),(10,D),(8,A),(9,C),(7,A),(13,D),(18,D)\rangle$	$\{G\}$	F
5	$\langle(0,-),(10,D),(8,A),(9,C),(7,A),(13,D),(16,F)\rangle$	$\{G\}$	F
8	$\langle(0,-),(10,D),(8,A),(9,C),(7,A),(13,D),(16,F)\rangle$	$\{\}$	G
5	$\langle(0,-),(10,D),(8,A),(9,C),(7,A),(13,D),(16,F)\rangle$	$\{\}$	G
12	$\langle(0,-),(10,D),(8,A),(9,C),(7,A),(13,D),(16,F)\rangle$	$\{\}$	-

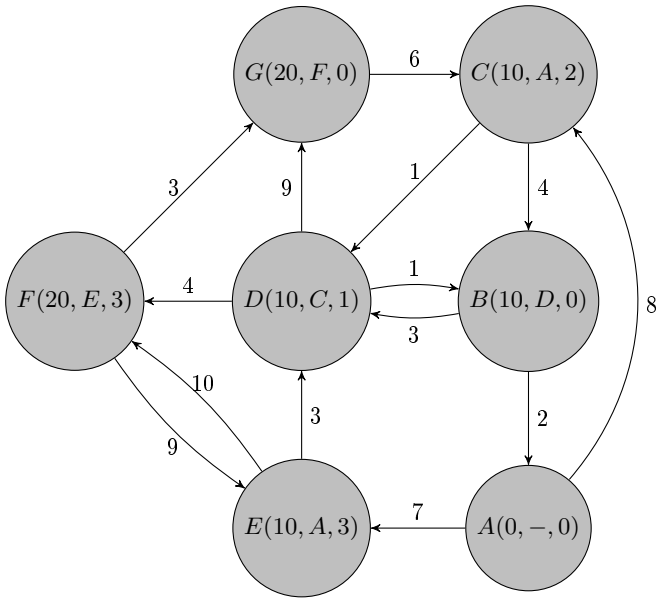
The shortest path from A to G is $A \to C \to D \to F \to G$, with a total weight of 16.

Item b

Mr Joaquim is now worried with the travel's cost. His car consumes 1 l per unit distance. The price of gasoline in cities $\{A, B, C, D, E, F, G\}$ is $\{1€, 2€, 1€, 2€, 1€, 1€, 1€\}$ respectively. Mr Joaquim only fills when needed (when he does not have enough gasoline to cross the road he wants to cross) and always fills 10 l of gasoline. Initially the car tank is empty. Implement an algorithm (in C++ or pseudocode) to determine the path Mr Joaquim should use so as to minimize the money spent on gasoline, and show the path he should follow. Explain. Suggestion: change Dijkstra's algorithm.

Algorithm 5 Dijkstra's algorithm

```
1:  function DIJKSTRA(G(V, E, w), price, s, d)
2:      for v ∈ V do cost(v) ← ∞, prev(v) ← NULL, tank(v) ← 0
3:      Q ← V
4:      cost(s) ← 0
5:      while |Q| > 0 do
6:          u ← vertex of Q with least dist(u)
7:          Q ← Q\{u}
8:          if u = d then break
9:          for v ∈ ADJ(G, u) do
10:             c' ← cost(u) + (w(u, v) ≤ tank(u)?0 : 10 * price(u))
11:             t' ← tank(u) + (w(u, v) ≤ tank(u)?0 : 10) − w(u, v)
12:             if c' < cost(v) or (c' = cost(v) and t' > tank(v)) then
13:                 cost(v) ← c', prev(v) ← u, tank(v) ← t'
14:     return dist, prev
```

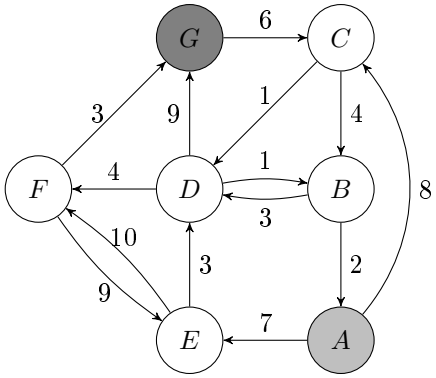


I am almost absolutely sure this algorithm is not optimal in these conditions, although I am not sure why it is not optimal.

The best path is $A \to E \to F \to G$. Mr Joaquim must refill his tank in A and E so he spends 20€.

Item c

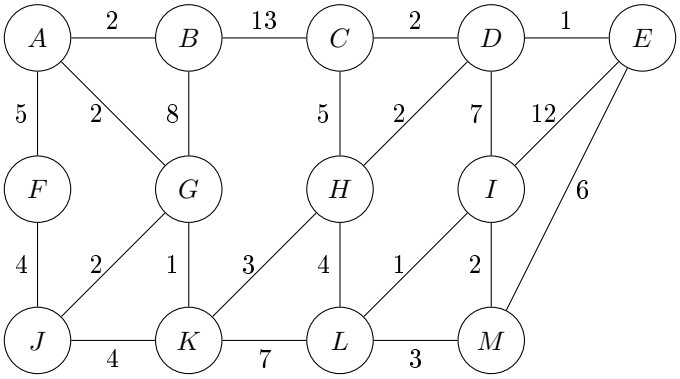
Mr Joaquim now wants to know/travel through all roads, but he does not want to go through any of them more than once. Is that possible? If so, show the path he should follow.



In white are vertices with in-degree equal to out-degree. In light gray are vertices with out-degree larger than in-degree by 1. In dark gray are vertices with in-degree larger than out-degree by 1. This means there is not an eulerian cycle because not all vertices have in-degree equal to out-degree, but there is an eulerian path starting in A and ending in G, which can be $A \to E \to F \to E \to D \to B \to A \to C \to B \to D \to F \to G \to C \to D \to G$.

Question 3

Consider the following undirected weighted graph G, where vertices represent places and an edge's value represents the cost associated to the connection between adjacent places.

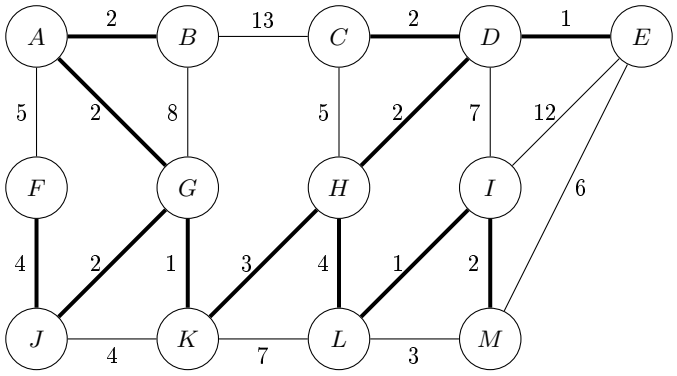


Item a

Find a minimum spanning tree, using the algorithm you find convenient. Present the order in which vertices are added to the minimum spanning tree, according to the algorithm you chose. Explain.

I will use Kruskal's algorithm. This algorithm consists of initially representing a graph as a forest of individual nodes; we order all edges by weight and process them by increasing weight, selecting edges that connect two separate trees until there is only one tree.

We will sort edges by increasing weight, then increasing first node and then increasing second node.



Edge	Weight	Action
D—*E*	1	Added
G—*K*	1	Added
I—*L*	1	Added
A—*B*	2	Added
A—*G*	2	Added
C—*D*	2	Added
D—*H*	2	Added
G—*J*	2	Added
I—*M*	2	Added
H—*K*	3	Added
L—*M*	3	Ignored
F—*J*	4	Added
H—*L*	4	Added
J—*K*	4	Ignored
A—*F*	5	Ignored
C—*H*	5	Ignored
E—*M*	6	Ignored
D—*I*	7	Ignored
K—*L*	7	Ignored
B—*G*	8	Ignored
E—*I*	12	Ignored
B—*C*	13	Ignored

Edges were added by the following order: *D*—*E*, *G*—*K*, *I*—*L*, *A*—*B*, *A*—*G*, *C*—*D*, *D*—*H*, *G*—*J*, *I*—*M*, *H*—*K*, *F*—*J*, *H*—*L*.

Item b

Implement an efficient algorithm (in pseudocode) that, knowing the minimum spanning tree T of a graph G, identifies the second minimum spanning tree (T_2). T_2 is the spanning tree of G having the least cost, other than T. Note that T and T_2 differ only by one edge.

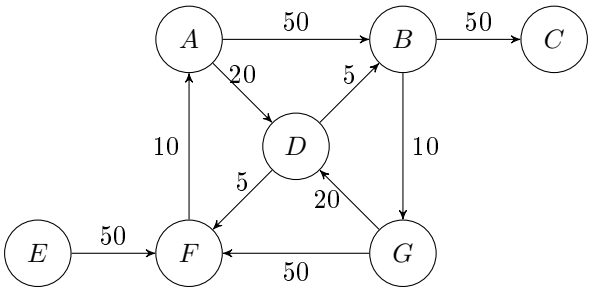
Diogo Miguel Ferreira Rodrigues (dmfrodrigues2000@gmail.com)

Algorithm 6 Second MST

```
1: function SecondMST(G(V, E, w), MST)
2:     res ← NULL, c ← ∞
3:     for (u, v) ∈ MST do
4:         MST' ← MST\(u, v)
5:         G' ← (V, MST', w)                          ▷ Has two tree components
6:         W' ← Σ_{(u',v')∈MST'} w(u', v')            ▷ Sum of all weights in G'
7:         S1 ← DFS(G', u)                            ▷ Tree component containing u
8:         S2 ← DFS(G', v)                            ▷ Tree component containing v
9:         for (u', v') ∈ E do
10:            if u' ∈ S1 and v' ∈ S2 and W' + w(u', v') < c then
11:                res ← MST' ∪ {(u', v')}, c ← W' + w(u', v')
12:    return res
```

Question 3

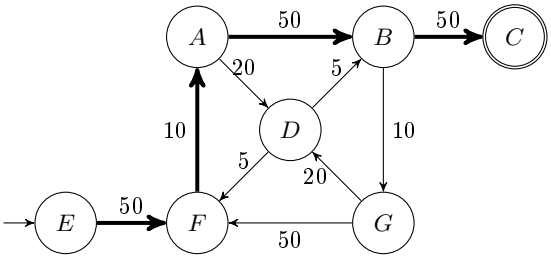
A drainage system was designed, according to the following graph, to transport 50,000 units of volume (uv) of waste water from the source in node E to the sink in node C. Transport is made by gravity, meaning canals have no pressure; in the network the canals' inclination is indicated with the direction of the arows, and their capacities in parenthesis are given in 1,000 uv. Answer the following questions, presenting the proper justification.

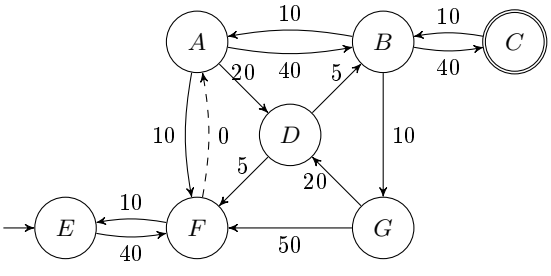


Item a

Can the network transport the desired volume of 50,000 uv? What is the maximum flow this network can transport?

We will use Edmonds-Karp algorithm to find the maximum flow from E to C. This algorithm operates by finding the shortest augmenting path in the residues graph and augment flow through that path, until there are no more augmenting paths. It specializes the Ford-Fulkerson algorithm by choosing the **shortest** (not *any*) augmenting path.





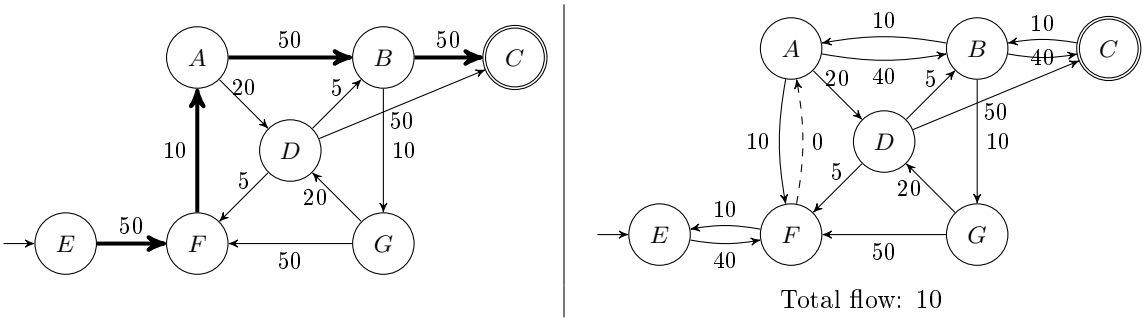
Total flow: 10

The maximum flow from E to C is thus 10,000 uv, so the network cannot transport the desired volume of 50,000 uv.

Diogo Miguel Ferreira Rodrigues (dmfrodrigues2000@gmail.com)

Item b

What happens to the maximum volume if a new sink, additionally to C, is placed in D?



Nothing happens, the maximum volume is similarly 10,000 uv. The bottleneck is edge $F \xrightarrow{10} A$.

Item c

What is the impact of reverting the flow direction (inclination) of canal GF (will become FG), in the network's performance, considering two consumers C and D?