Tutorial Problem
Real Estate Market

Daniel Graf

ETH Zürich

December 6, 2017

Problem Statement

Input describing a big property sale:

- N bidders for M locations that each belong to one of S states
- $B = (b_{i,j})$, the bid of bidder i for location j
- $\ell = (\ell_1, \ldots, \ell_S)$: limit on number of locations sold per state
- $s = (s_1, \ldots, s_M)$: locations to state assignments

Question: How many sites can be sold and what is the maximum profit?

Example 1:

- $N = 3$, $M = 3$, $S = 1$
- $\ell = (3)$, $s = (1, 1, 1)$
- $B = \begin{pmatrix} 7 & 7 & 8 \\ 2 & 9 & 3 \\ 5 & 2 & 4 \end{pmatrix}$

Example 2:

- $N = 3$, $M = 3$, $S = 1$
- $\ell = (2)$, $s = (1, 1, 1)$
- $B = \begin{pmatrix} 7 & 7 & 8 \\ 2 & 9 & 3 \\ 5 & 2 & 4 \end{pmatrix}$

Example 3:

- $N = 3$, $M = 3$, $S = 2$
- $\ell = (1, 1)$, $s = (1, 2, 2)$
- $B = \begin{pmatrix} 7 & 7 & 8 \\ 2 & 9 & 3 \\ 5 & 2 & 4 \end{pmatrix}$

Problem Statement

Input describing a big property sale:

- N bidders for M locations that each belong to one of S states
- $B = (b_{i,j})$, the bid of bidder i for location j
- $\ell = (\ell_1, \dots, \ell_S)$: limit on number of locations sold per state
- $s = (s_1, \dots, s_M)$: locations to state assignments

Question: How many sites can be sold and what is the maximum profit?

Example 1:

- $N = 3$, $M = 3$, $S = 1$
- $\ell = (3)$, $s = (1, 1, 1)$
- $B = \begin{pmatrix} 7 & 7 & 8 \\ 2 & 9 & 3 \\ 5 & 2 & 4 \end{pmatrix}$

Example 2:

- $N = 3$, $M = 3$, $S = 1$
- $\ell = (2)$, $s = (1, 1, 1)$
- $B = \begin{pmatrix} 7 & 7 & 8 \\ 2 & 9 & 3 \\ 5 & 2 & 4 \end{pmatrix}$

Example 3:

- $N = 3$, $M = 3$, $S = 2$
- $\ell = (1, 1)$, $s = (1, 2, 2)$
- $B = \begin{pmatrix} 7 & 7 & 8 \\ 2 & 9 & 3 \\ 5 & 2 & 4 \end{pmatrix}$

Problem Statement

Input describing a big property sale:

- N bidders for M locations that each belong to one of S states
- $B = (b_{i,j})$, the bid of bidder i for location j
- $\ell = (\ell_1, \ldots, \ell_S)$: limit on number of locations sold per state
- $s = (s_1, \ldots, s_M)$: locations to state assignments

Question: How many sites can be sold and what is the maximum profit?

Example 1:

- $N = 3,\ M = 3,\ S = 1$
- $\ell = (3),\ s = (1, 1, 1)$
- $B = \begin{pmatrix} 7 & 7 & 8 \\ 2 & 9 & 3 \\ 5 & 2 & 4 \end{pmatrix}$

Example 2:

- $N = 3,\ M = 3,\ S = 1$
- $\ell = (2),\ s = (1, 1, 1)$
- $B = \begin{pmatrix} 7 & 7 & 8 \\ 2 & 9 & 3 \\ 5 & 2 & 4 \end{pmatrix}$

Example 3:

- $N = 3,\ M = 3,\ S = 2$
- $\ell = (1, 1),\ s = (1, 2, 2)$
- $B = \begin{pmatrix} 7 & 7 & 8 \\ 2 & 9 & 3 \\ 5 & 2 & 4 \end{pmatrix}$

Problem Statement

Input describing a big property sale:

- N bidders for M locations that each belong to one of S states
- $B = (b_{i,j})$, the bid of bidder i for location j
- $\ell = (\ell_1, \dots, \ell_S)$: limit on number of locations sold per state
- $s = (s_1, \dots, s_M)$: locations to state assignments

Question: How many sites can be sold and what is the maximum profit?

Example 1:

- $N = 3$, $M = 3$, $S = 1$
- $\ell = (3)$, $s = (1, 1, 1)$
- $B = \begin{pmatrix} 7 & 7 & 8 \\ 2 & 9 & 3 \\ 5 & 2 & 4 \end{pmatrix}$

Example 2:

- $N = 3$, $M = 3$, $S = 1$
- $\ell = (2)$, $s = (1, 1, 1)$
- $B = \begin{pmatrix} 7 & 7 & 8 \\ 2 & 9 & 3 \\ 5 & 2 & 4 \end{pmatrix}$

Example 3:

- $N = 3$, $M = 3$, $S = 2$
- $\ell = (1, 1)$, $s = (1, 2, 2)$
- $B = \begin{pmatrix} 7 & 7 & 8 \\ 2 & 9 & 3 \\ 5 & 2 & 4 \end{pmatrix}$

First Steps

Look at the limits!

- $N, M \leq 100$
- Bruteforcing all subsets is too slow.
- Greedy will probably not work.
- Can we formulate it as an LP?
  Or as a flow problem?

First Attempt: Linear Program

We formulate it as an integer linear program:

- $N \cdot M$ variables: $x_{i,j} \in \{0,1\}$
  $x_{i,j} = 1$ if and only if bidder i gets location j.
- Objective function: maximize $\sum_{i,j} x_{i,j} \cdot b_{i,j}$.
- Constraints:
  - At most one location per buyer:
    $\forall i$: $\sum_j x_{i,j} \leq 1$.
  - At most one buyer per location:
    $\forall j$: $\sum_i x_{i,j} \leq 1$.
  - At most l_i sales per state:
    $\forall k$: $\sum_{j \in \{j | s_j = k\}} \sum_i x_{i,j} \leq \ell_k$.

Relax it to a linear program: $x_{i,j} \in [0,1]$.

Will it work?

- It is not obvious why this relaxation works, i.e. why there is always an integer solution that achieves the maximum profit of the LP. (But it does as we will see.)

Will it be fast enough?

- $N \cdot M = 10'000$ variables.
  $\Rightarrow$ It will be too slow.
  ($\approx$ 1 minute per testcase.)

So let us try something else.

First Attempt: Linear Program

We formulate it as an integer linear program:

- $N \cdot M$ variables: $x_{i,j} \in \{0, 1\}$
  $x_{i,j} = 1$ if and only if bidder i gets location j.
- Objective function: maximize $\sum_{i,j} x_{i,j} \cdot b_{i,j}$.
- Constraints:
  - At most one location per buyer:
    $\forall i$: $\sum_j x_{i,j} \leq 1$.
  - At most one buyer per location:
    $\forall j$: $\sum_i x_{i,j} \leq 1$.
  - At most l_i sales per state:
    $\forall k$: $\sum_{j \in \{j | s_j = k\}} \sum_i x_{i,j} \leq \ell_k$.

Relax it to a linear program: $x_{i,j} \in [0, 1]$.

Will it work?

- It is not obvious why this relaxation works, i.e. why there is always an integer solution that achieves the maximum profit of the LP.
  (But it does as we will see.)

Will it be fast enough?

- $N \cdot M = 10'000$ variables.
  $\Rightarrow$ It will be too slow.
  ($\approx$ 1 minute per testcase.)

So let us try something else.

First Attempt: Linear Program

We formulate it as an integer linear program:

- $N \cdot M$ variables: $x_{i,j} \in \{0, 1\}$
  $x_{i,j} = 1$ if and only if bidder i gets location j.
- Objective function: maximize $\sum_{i,j} x_{i,j} \cdot b_{i,j}$.
- Constraints:
  - At most one location per buyer:
    $\forall i$: $\sum_j x_{i,j} \leq 1$.
  - At most one buyer per location:
    $\forall j$: $\sum_i x_{i,j} \leq 1$.
  - At most l_i sales per state:
    $\forall k$: $\sum_{j \in \{j | s_j = k\}} \sum_i x_{i,j} \leq \ell_k$.

Relax it to a linear program: $x_{i,j} \in [0, 1]$.

Will it work?

- It is not obvious why this relaxation works, i.e. why there is always an integer solution that achieves the maximum profit of the LP. (But it does as we will see.)

Will it be fast enough?

- $N \cdot M = 10'000$ variables.
  $\Rightarrow$ It will be too slow.
  (≈ 1 minute per testcase.)

So let us try something else.

First Attempt: Linear Program

We formulate it as an integer linear program:

- $N \cdot M$ variables: $x_{i,j} \in \{0, 1\}$
  $x_{i,j} = 1$ if and only if bidder i gets location j.
- Objective function: maximize $\sum_{i,j} x_{i,j} \cdot b_{i,j}$.
- Constraints:
  - At most one location per buyer:
    $\forall i$: $\sum_j x_{i,j} \leq 1$.
  - At most one buyer per location:
    $\forall j$: $\sum_i x_{i,j} \leq 1$.
  - At most l_i sales per state:
    $\forall k$: $\sum_{j \in \{j | s_j = k\}} \sum_i x_{i,j} \leq \ell_k$.

Relax it to a linear program: $x_{i,j} \in [0, 1]$.

Will it work?

- It is not obvious why this relaxation works, i.e. why there is always an integer solution that achieves the maximum profit of the LP. (But it does as we will see.)

Will it be fast enough?

- $N \cdot M = 10'000$ variables.
  $\Rightarrow$ It will be too slow.
  ($\approx$ 1 minute per testcase.)

So let us try something else.

First Attempt: Linear Program

We formulate it as an integer linear program:

- $N \cdot M$ variables: $x_{i,j} \in \{0, 1\}$
  $x_{i,j} = 1$ if and only if bidder i gets location j.
- Objective function: maximize $\sum_{i,j} x_{i,j} \cdot b_{i,j}$.
- Constraints:
  - At most one location per buyer:
    $\forall i$: $\sum_j x_{i,j} \leq 1$.
  - At most one buyer per location:
    $\forall j$: $\sum_i x_{i,j} \leq 1$.
  - At most l_i sales per state:
    $\forall k$: $\sum_{j \in \{j | s_j = k\}} \sum_i x_{i,j} \leq \ell_k$.

Relax it to a linear program: $x_{i,j} \in [0, 1]$.

Will it work?

- It is not obvious why this relaxation works, i.e. why there is always an integer solution that achieves the maximum profit of the LP.
  (But it does as we will see.)

Will it be fast enough?

- $N \cdot M = 10'000$ variables.
  $\Rightarrow$ It will be too slow.
  ($\approx$ 1 minute per testcase.)

So let us try something else.

First Attempt: Linear Program

We formulate it as an integer linear program:

- $N \cdot M$ variables: $x_{i,j} \in \{0, 1\}$
  $x_{i,j} = 1$ if and only if bidder i gets location j.
- Objective function: maximize $\sum_{i,j} x_{i,j} \cdot b_{i,j}$.
- Constraints:
  - At most one location per buyer:
    $\forall i$: $\sum_j x_{i,j} \leq 1$.
  - At most one buyer per location:
    $\forall j$: $\sum_i x_{i,j} \leq 1$.
  - At most l_i sales per state:
    $\forall k$: $\sum_{j \in \{j | s_j = k\}} \sum_i x_{i,j} \leq \ell_k$.

Relax it to a linear program: $x_{i,j} \in [0, 1]$.

Will it work?

- It is not obvious why this relaxation works, i.e. why there is always an integer solution that achieves the maximum profit of the LP. (But it does as we will see.)

Will it be fast enough?

- $N \cdot M = 10'000$ variables.
  $\Rightarrow$ It will be too slow.
  ($\approx$ 1 minute per testcase.)

So let us try something else.

First Attempt: Linear Program

We formulate it as an integer linear program:

- $N \cdot M$ variables: $x_{i,j} \in \{0, 1\}$
  $x_{i,j} = 1$ if and only if bidder i gets location j.
- Objective function: maximize $\sum_{i,j} x_{i,j} \cdot b_{i,j}$.
- Constraints:
  - At most one location per buyer:
    $\forall i$: $\sum_j x_{i,j} \leq 1$.
  - At most one buyer per location:
    $\forall j$: $\sum_i x_{i,j} \leq 1$.
  - At most l_i sales per state:
    $\forall k$: $\sum_{j \in \{j \mid s_j = k\}} \sum_i x_{i,j} \leq \ell_k$.

Relax it to a linear program: $x_{i,j} \in [0, 1]$.

Will it work?

- It is not obvious why this relaxation works, i.e. why there is always an integer solution that achieves the maximum profit of the LP. (But it does as we will see.)

Will it be fast enough?

- $N \cdot M = 10'000$ variables.
  $\Rightarrow$ It will be too slow.
  ($\approx$ 1 minute per testcase.)

So let us try something else.

First Attempt: Linear Program

We formulate it as an integer linear program:

- $N \cdot M$ variables: $x_{i,j} \in \{0, 1\}$
  $x_{i,j} = 1$ if and only if bidder i gets location j.
- Objective function: maximize $\sum_{i,j} x_{i,j} \cdot b_{i,j}$.
- Constraints:
  - At most one location per buyer:
    $\forall i$: $\sum_j x_{i,j} \leq 1$.
  - At most one buyer per location:
    $\forall j$: $\sum_i x_{i,j} \leq 1$.
  - At most l_i sales per state:
    $\forall k$: $\sum_{j \in \{j | s_j = k\}} \sum_i x_{i,j} \leq \ell_k$.

Relax it to a linear program: $x_{i,j} \in [0, 1]$.

Will it work?

- It is not obvious why this relaxation works, i.e. why there is always an integer solution that achieves the maximum profit of the LP. (But it does as we will see.)

Will it be fast enough?

- $N \cdot M = 10'000$ variables.
  $\Rightarrow$ It will be too slow.
  ($\approx$ 1 minute per testcase.)

So let us try something else.

First Attempt: Linear Program

We formulate it as an integer linear program:

- $N \cdot M$ variables: $x_{i,j} \in \{0, 1\}$
  $x_{i,j} = 1$ if and only if bidder i gets location j.
- Objective function: maximize $\sum_{i,j} x_{i,j} \cdot b_{i,j}$.
- Constraints:
  - At most one location per buyer:
    $\forall i$: $\sum_j x_{i,j} \leq 1$.
  - At most one buyer per location:
    $\forall j$: $\sum_i x_{i,j} \leq 1$.
  - At most l_i sales per state:
    $\forall k$: $\sum_{j \in \{j | s_j = k\}} \sum_i x_{i,j} \leq \ell_k$.

Relax it to a linear program: $x_{i,j} \in [0, 1]$.

Will it work?

- It is not obvious why this relaxation works, i.e. why there is always an integer solution that achieves the maximum profit of the LP. (But it does as we will see.)

Will it be fast enough?

- $N \cdot M = 10'000$ variables.
  $\Rightarrow$ It will be too slow.
  ($\approx$ 1 minute per testcase.)

So let us try something else.

Second Attempt: Min Cost Max Flow

Look at the first subtask!

- $N = M$, $S = 1$, $\ell_1 = N$
  There is a location for every bidder.
  No other constraints from the states.
- $1 \leq b_{i,j} \leq 100$
  No negative or zero bids.
  If we can sell more, we do sell more.

$\Rightarrow$ We will sell all M locations and "just" need to find the permutation that assigns the bidders to locations in the optimal way.

Apply pattern matching!
What does this task remind you of?

- Very similar to the fruit delivery problem we saw in class.
- Only differences: supplies and demands are all 1 and maximum profit instead of minimum cost
- Formally: maximum weight perfect matching in a bipartite graph

$\Rightarrow$ Rephrase it as a MaxCostMaxFlow.

Second Attempt: Min Cost Max Flow

Look at the first subtask!

- $N = M$, $S = 1$, $\ell_1 = N$
  There is a location for every bidder.
  No other constraints from the states.
- $1 \leq b_{i,j} \leq 100$
  No negative or zero bids.
  If we can sell more, we do sell more.

$\Rightarrow$ We will sell all M locations and "just" need to find the permutation that assigns the bidders to locations in the optimal way.

Apply pattern matching!
What does this task remind you of?

- Very similar to the fruit delivery problem we saw in class.
- Only differences: supplies and demands are all 1 and maximum profit instead of minimum cost
- Formally: maximum weight perfect matching in a bipartite graph

$\Rightarrow$ Rephrase it as a MaxCostMaxFlow.

Second Attempt: Min Cost Max Flow

Look at the first subtask!

- $N = M$, $S = 1$, $\ell_1 = N$
  There is a location for every bidder.
  No other constraints from the states.
- $1 \leq b_{i,j} \leq 100$
  No negative or zero bids.
  If we can sell more, we do sell more.

$\Rightarrow$ We will sell all M locations and "just" need to find the permutation that assigns the bidders to locations in the optimal way.

Apply pattern matching!
What does this task remind you of?

- Very similar to the fruit delivery problem we saw in class.
- Only differences: supplies and demands are all 1 and maximum profit instead of minimum cost
- Formally: maximum weight perfect matching in a bipartite graph

$\Rightarrow$ Rephrase it as a MaxCostMaxFlow.

Second Attempt: Min Cost Max Flow

Look at the first subtask!

- $N = M$, $S = 1$, $\ell_1 = N$
  There is a location for every bidder.
  No other constraints from the states.
- $1 \leq b_{i,j} \leq 100$
  No negative or zero bids.
  If we can sell more, we do sell more.

$\Rightarrow$ We will sell all M locations and "just" need to find the permutation that assigns the bidders to locations in the optimal way.

Apply pattern matching!
What does this task remind you of?

- Very similar to the fruit delivery problem we saw in class.
- Only differences: supplies and demands are all 1 and maximum profit instead of minimum cost
- Formally: maximum weight perfect matching in a bipartite graph

$\Rightarrow$ Rephrase it as a MaxCostMaxFlow.

Second Attempt: Min Cost Max Flow

Look at the first subtask!

- $N = M$, $S = 1$, $\ell_1 = N$
  There is a location for every bidder.
  No other constraints from the states.
- $1 \leq b_{i,j} \leq 100$
  No negative or zero bids.
  If we can sell more, we do sell more.

$\Rightarrow$ We will sell all M locations and "just" need to find the permutation that assigns the bidders to locations in the optimal way.

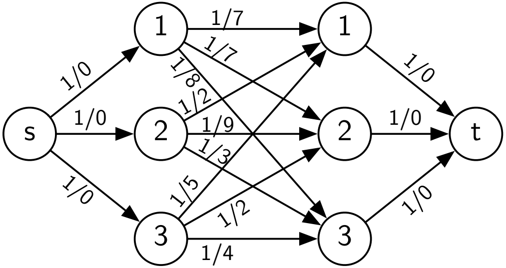
Apply pattern matching!
What does this task remind you of?

- Very similar to the fruit delivery problem we saw in class.
- Only differences: supplies and demands are all 1 and maximum profit instead of minimum cost
- Formally: maximum weight perfect matching in a bipartite graph

$\Rightarrow$ Rephrase it as a MaxCostMaxFlow.

Second Attempt: Min Cost Max Flow

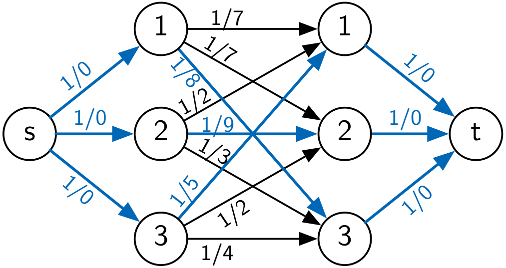
$$\begin{pmatrix} 7 & 7 & 8 \\ 2 & 9 & 3 \\ 5 & 2 & 4 \end{pmatrix} \Rightarrow$$



- Negate all the costs to get a Min Cost Max Flow problem.
- Use `cycle_canceling()` and get the 20 points for this subtask.

Second Attempt: Min Cost Max Flow

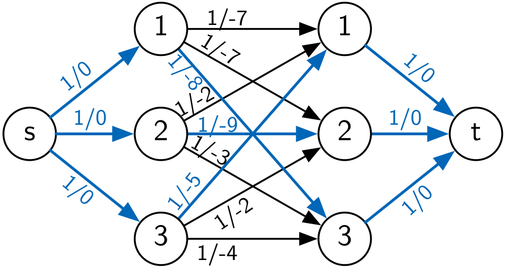
$$\begin{pmatrix} 7 & 7 & 8 \\ 2 & 9 & 3 \\ 5 & 2 & 4 \end{pmatrix} \Rightarrow$$



- Negate all the costs to get a Min Cost Max Flow problem.
- Use `cycle_canceling()` and get the 20 points for this subtask.

Second Attempt: Min Cost Max Flow

$$\begin{pmatrix} 7 & 7 & 8 \\ 2 & 9 & 3 \\ 5 & 2 & 4 \end{pmatrix} \Rightarrow$$



- Negate all the costs to get a Min Cost Max Flow problem.
- Use `cycle_canceling()` and get the 20 points for this subtask.

Next Subtasks

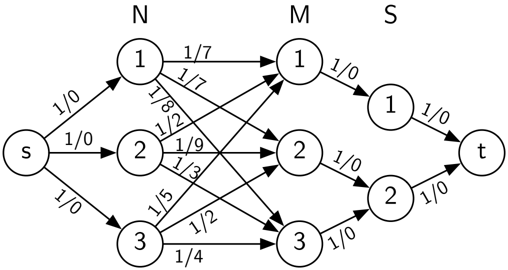
How to incorporate the state legislations?

- Add an extra layer of vertices to the graph to limit the flow per state.

$$N = 3, M = 3, S = 2$$

$$\ell = (1, 1), s = (1, 2, 2)$$

$$B = \begin{pmatrix} 7 & 7 & 8 \\ 2 & 9 & 3 \\ 5 & 2 & 4 \end{pmatrix} \Rightarrow$$



If you do this and submit, you will get the verdict CORRECT for the first four test cases and so score at least 80 points.
(And it also explains why the LP-solution works: Flow Integrality Theorem.)

Next Subtasks

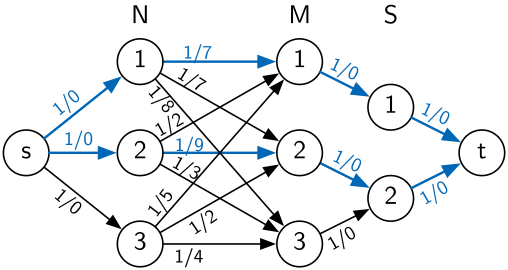
How to incorporate the state legislations?

- Add an extra layer of vertices to the graph to limit the flow per state.

$N = 3, M = 3, S = 2$

$\ell = (1, 1), s = (1, 2, 2)$

$$B = \begin{pmatrix} 7 & 7 & 8 \\ 2 & 9 & 3 \\ 5 & 2 & 4 \end{pmatrix} \Rightarrow$$



If you do this and submit, you will get the verdict CORRECT for the first four test cases and so score at least 80 points.
(And it also explains why the LP-solution works: Flow Integrality Theorem.)

Next Subtasks

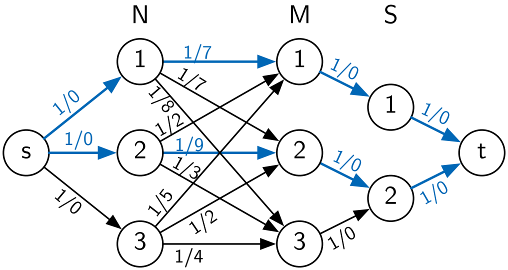
How to incorporate the state legislations?

- Add an extra layer of vertices to the graph to limit the flow per state.

$N = 3, M = 3, S = 2$

$\ell = (1, 1), s = (1, 2, 2)$

$$B = \begin{pmatrix} 7 & 7 & 8 \\ 2 & 9 & 3 \\ 5 & 2 & 4 \end{pmatrix} \Rightarrow$$



If you do this and submit, you will get the verdict CORRECT for the first four test cases and so score at least 80 points.

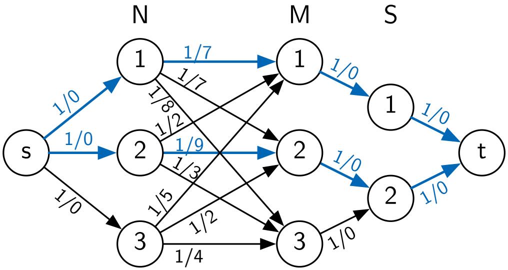
(And it also explains why the LP-solution works: Flow Integrality Theorem.)

Next Subtasks

How to incorporate the state legislations?

- Add an extra layer of vertices to the graph to limit the flow per state.

$$N = 3, M = 3, S = 2$$
$$\ell = (1,1), s = (1,2,2)$$
$$B = \begin{pmatrix} 7 & 7 & 8 \\ 2 & 9 & 3 \\ 5 & 2 & 4 \end{pmatrix} \Rightarrow$$

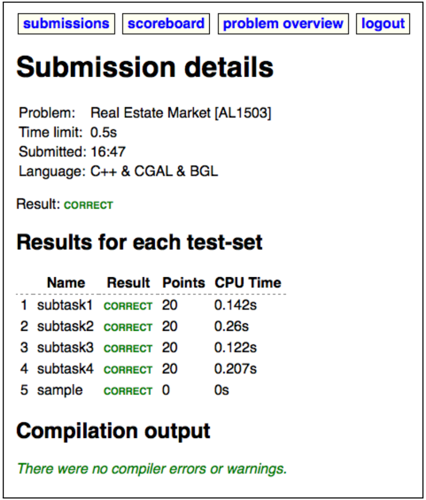


If you do this and submit, you will get the verdict CORRECT for the first four test cases and so score at least 80 points.
(And it also explains why the LP-solution works: Flow Integrality Theorem.)

Are we done?

Will we get 100 points? How can we find out before looking at the verdict of the last subtask?

- Our code takes 0.2s for subtask 4 where $N \cdot M \leq 10^3$ holds.
- Timelimit is 0.5s and subtask 5 has limits of $N \cdot M \leq 10^4$.
- Our code will be at least 10x slower on subtask 5, so it will be too slow.



submissions | scoreboard | problem overview | logout

Submission details

Problem: Real Estate Market [AL1503]
Time limit: 0.5s
Submitted: 16:47
Language: C++ & CGAL & BGL

Result: CORRECT

Results for each test-set

	Name	Result	Points	CPU Time
1	subtask1	CORRECT	20	0.142s
2	subtask2	CORRECT	20	0.26s
3	subtask3	CORRECT	20	0.122s
4	subtask4	CORRECT	20	0.207s
5	sample	CORRECT	0	0s

Compilation output

There were no compiler errors or warnings.

Getting full score

Can we use a faster algorithm or do we need a fundamentally different solution?

- `successive_shortest_path_nonnegative_weights()`
  faster but requires non-negative weights
- Next questions: How can we use it and how much faster will it be?

How much faster will it be? Back-of-the-envelope calculations never hurt.

- Recall the runtime bounds from the documentation:
  - `cycle_canceling()`: $\mathcal{O}(C \cdot (nm))$
  - `successive_shortest_path_nonnegative_weights()`: $\mathcal{O}(|f| \cdot (m + n \log n))$
- Estimate the variables:
  - $n = N + M + S + 2 \approx 300$, $m = N \cdot M + N + M + S \approx 10'000$.
  - $C \leq \max b_{i,j} \cdot \min(N, M) \leq 10'000$, $|f| \leq \min(N, M) \leq 100$
- Get the rough estimates:
  - `cycle_canceling()`: Order of $10^{10} \Rightarrow$ too slow
  - `successive_shortest_path_nonnegative_weights()`: Order of $10^{6} \Rightarrow$ ok

Getting full score

Can we use a faster algorithm or do we need a fundamentally different solution?

- `successive_shortest_path_nonnegative_weights()`
  faster but requires non-negative weights
- Next questions: How can we use it and how much faster will it be?

How much faster will it be? Back-of-the-envelope calculations never hurt.

- Recall the runtime bounds from the documentation:
  - `cycle_canceling()`: $\mathcal{O}(C \cdot (nm))$
  - `successive_shortest_path_nonnegative_weights()`: $\mathcal{O}(|f| \cdot (m + n \log n))$
- Estimate the variables:
  - $n = N + M + S + 2 \approx 300$, $m = N \cdot M + N + M + S \approx 10'000$.
  - $C \leq \max b_{i,j} \cdot \min(N, M) \leq 10'000$, $|f| \leq \min(N, M) \leq 100$
- Get the rough estimates:
  - `cycle_canceling()`: Order of 10^{10} $\Rightarrow$ too slow
  - `successive_shortest_path_nonnegative_weights()`: Order of 10^6 $\Rightarrow$ ok

Getting full score

Can we use a faster algorithm or do we need a fundamentally different solution?

- `successive_shortest_path_nonnegative_weights()`
  faster but requires non-negative weights
- Next questions: How can we use it and how much faster will it be?

How much faster will it be? Back-of-the-envelope calculations never hurt.

- Recall the runtime bounds from the documentation:
  - `cycle_canceling()`: $\mathcal{O}(C \cdot (nm))$
  - `successive_shortest_path_nonnegative_weights()`: $\mathcal{O}(|f| \cdot (m + n \log n))$
- Estimate the variables:
  - $n = N + M + S + 2 \approx 300$, $m = N \cdot M + N + M + S \approx 10'000$.
  - $C \leq \max b_{i,j} \cdot \min(N, M) \leq 10'000$, $|f| \leq \min(N, M) \leq 100$
- Get the rough estimates:
  - `cycle_canceling()`: Order of 10^{10} $\Rightarrow$ too slow
  - `successive_shortest_path_nonnegative_weights()`: Order of 10^{6} $\Rightarrow$ ok

Getting full score

Can we use a faster algorithm or do we need a fundamentally different solution?

- `successive_shortest_path_nonnegative_weights()`
  faster but requires non-negative weights
- Next questions: How can we use it and how much faster will it be?

How much faster will it be? Back-of-the-envelope calculations never hurt.

- Recall the runtime bounds from the documentation:
  - `cycle_canceling()`: $\mathcal{O}(C \cdot (nm))$
  - `successive_shortest_path_nonnegative_weights()`: $\mathcal{O}(|f| \cdot (m + n \log n))$
- Estimate the variables:
  - $n = N + M + S + 2 \approx 300$, $m = N \cdot M + N + M + S \approx 10'000$.
  - $C \leq \max b_{i,j} \cdot \min(N, M) \leq 10'000$, $|f| \leq \min(N, M) \leq 100$
- Get the rough estimates:
  - `cycle_canceling()`: Order of 10^{10} $\Rightarrow$ too slow
  - `successive_shortest_path_nonnegative_weights()`: Order of 10^6 $\Rightarrow$ ok

Getting full score

Can we use a faster algorithm or do we need a fundamentally different solution?

- `successive_shortest_path_nonnegative_weights()`
  faster but requires non-negative weights
- Next questions: How can we use it and how much faster will it be?

How much faster will it be? Back-of-the-envelope calculations never hurt.

- Recall the runtime bounds from the documentation:
  - `cycle_canceling()`: $\mathcal{O}(C \cdot (nm))$
  - `successive_shortest_path_nonnegative_weights()`: $\mathcal{O}(|f| \cdot (m + n \log n))$
- Estimate the variables:
  - $n = N + M + S + 2 \approx 300$, $m = N \cdot M + N + M + S \approx 10'000$.
  - $C \leq \max b_{i,j} \cdot \min(N, M) \leq 10'000$, $|f| \leq \min(N, M) \leq 100$
- Get the rough estimates:
  - `cycle_canceling()`: Order of 10^{10} $\Rightarrow$ too slow
  - `successive_shortest_path_nonnegative_weights()`: Order of 10^6 $\Rightarrow$ ok

Getting full score

Can we use a faster algorithm or do we need a fundamentally different solution?

- `successive_shortest_path_nonnegative_weights()`
  faster but requires non-negative weights
- Next questions: How can we use it and how much faster will it be?

How much faster will it be? Back-of-the-envelope calculations never hurt.

- Recall the runtime bounds from the documentation:
  - `cycle_canceling()`: $\mathcal{O}(C \cdot (nm))$
  - `successive_shortest_path_nonnegative_weights()`: $\mathcal{O}(|f| \cdot (m + n \log n))$
- Estimate the variables:
  - $n = N + M + S + 2 \approx 300$, $m = N \cdot M + N + M + S \approx 10'000$.
  - $C \leq \max b_{i,j} \cdot \min(N, M) \leq 10'000$, $|f| \leq \min(N, M) \leq 100$
- Get the rough estimates:
  - `cycle_canceling()`: Order of $10^{10} \Rightarrow$ too slow
  - `successive_shortest_path_nonnegative_weights()`: Order of $10^6 \Rightarrow$ ok

Getting full score

Can we use a faster algorithm or do we need a fundamentally different solution?

- `successive_shortest_path_nonnegative_weights()`
  faster but requires non-negative weights
- Next questions: How can we use it and how much faster will it be?

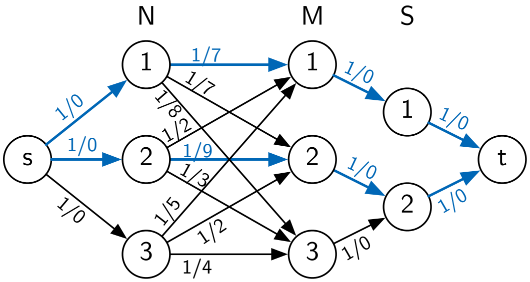
How much faster will it be? Back-of-the-envelope calculations never hurt.

- Recall the runtime bounds from the documentation:
  - `cycle_canceling()`: $\mathcal{O}(C \cdot (nm))$
  - `successive_shortest_path_nonnegative_weights()`: $\mathcal{O}(|f| \cdot (m + n \log n))$
- Estimate the variables:
  - $n = N + M + S + 2 \approx 300$, $m = N \cdot M + N + M + S \approx 10'000$.
  - $C \leq \max b_{i,j} \cdot \min(N, M) \leq 10'000$, $|f| \leq \min(N, M) \leq 100$
- Get the rough estimates:
  - `cycle_canceling()`: Order of 10^{10} $\Rightarrow$ too slow
  - `successive_shortest_path_nonnegative_weights()`: Order of 10^{6} $\Rightarrow$ ok

Getting Full Score

How can we use it? How can we eliminate the negative weights?

- What if we add a constant Δ to all the edges between bidders and locations?

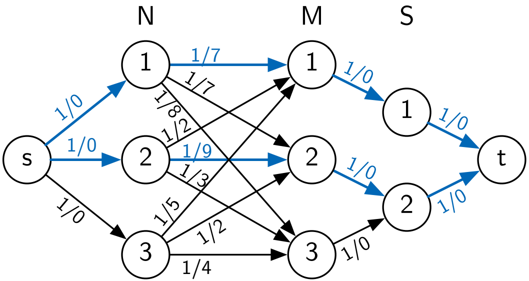


- Every unit of flow uses exactly one edge that got more expensive.
- Total cost increases by $|f| \cdot \Delta$ and the optimum matching does not change.
- If we set $\Delta = \max b_{i,j}$ or $\Delta = 100$, we get non-negative costs.
- For the final result, we just subtract $|f| \cdot \Delta$ from the costs that we got.

Getting Full Score

How can we use it? How can we eliminate the negative weights?

- What if we add a constant Δ to all the edges between bidders and locations?

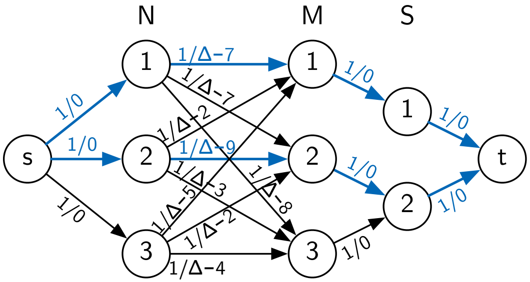


- Every unit of flow uses exactly one edge that got more expensive.
- Total cost increases by $|f| \cdot \Delta$ and the optimum matching does not change.
- If we set $\Delta = \max b_{i,j}$ or $\Delta = 100$, we get non-negative costs.
- For the final result, we just subtract $|f| \cdot \Delta$ from the costs that we got.

Getting Full Score

How can we use it? How can we eliminate the negative weights?

- What if we add a constant Δ to all the edges between bidders and locations?

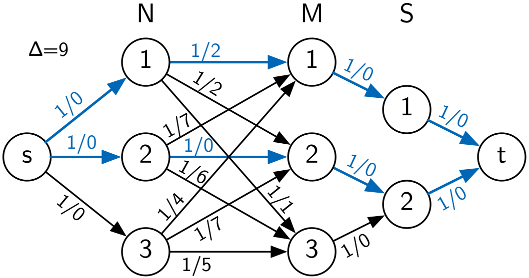


- Every unit of flow uses exactly one edge that got more expensive.
- Total cost increases by $|f| \cdot \Delta$ and the optimum matching does not change.
- If we set $\Delta = \max b_{i,j}$ or $\Delta = 100$, we get non-negative costs.
- For the final result, we just subtract $|f| \cdot \Delta$ from the costs that we got.

Getting Full Score

How can we use it? How can we eliminate the negative weights?

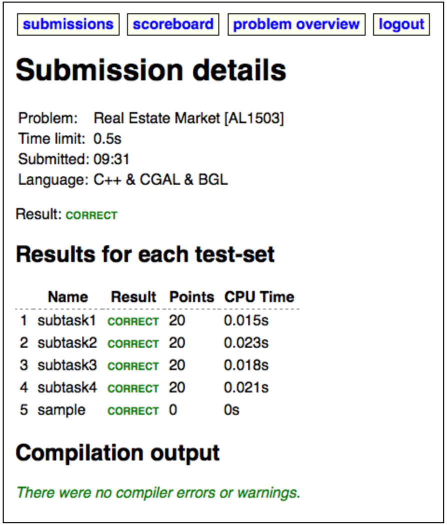
- What if we add a constant Δ to all the edges between bidders and locations?



- Every unit of flow uses exactly one edge that got more expensive.
- Total cost increases by $|f| \cdot \Delta$ and the optimum matching does not change.
- If we set $\Delta = \max b_{i,j}$ or $\Delta = 100$, we get non-negative costs.
- For the final result, we just subtract $|f| \cdot \Delta$ from the costs that we got.

Getting Full Score

If we submit again, we see that our program got roughly 10x faster and we have factor 25x left for the last subtask.



submissions | scoreboard | problem overview | logout

Submission details

Problem: Real Estate Market [AL1503]
Time limit: 0.5s
Submitted: 09:31
Language: C++ & CGAL & BGL

Result: CORRECT

Results for each test-set

	Name	Result	Points	CPU Time
1	subtask1	CORRECT	20	0.015s
2	subtask2	CORRECT	20	0.023s
3	subtask3	CORRECT	20	0.018s
4	subtask4	CORRECT	20	0.021s
5	sample	CORRECT	0	0s

Compilation output

There were no compiler errors or warnings.

General Tips and Tricks

Use the public test sets, run and time your solution on them before you submit.

- Pipe input/output from the files: (no need for copy/pasting)

```
$ time ./program < input.in > youroutput.out
    real   0m0.349s
    user   0m0.335s
    sys 0m0.009s
```

- Use `diff` to check for mistakes: (no output = no mistake)

```
$ diff output.out youroutput.out
    1c1
    < 21
    ---
    > 42
```

- One line to do it all: (quick way of running it on all the samples before submitting)

```
$ for f in *.in; do echo "-- $f --"; time ./program < $f > ${f
    ↪ %.*}.myout; diff ${f%.*}.out ${f%.*}.myout; done
```