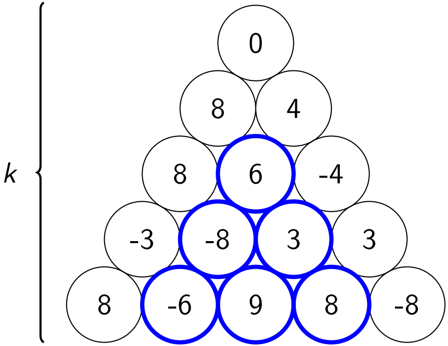
Threefold Problem Set #4
Christmas Presents: Solutions

December 14, 2022

Alice's Accumulation – Naive Solution



Goal: maximize the sum of a subpile.

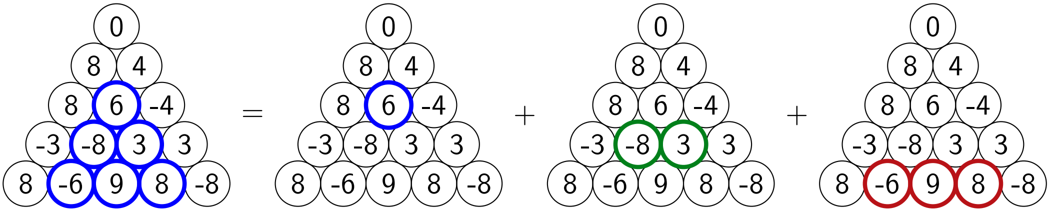
Naive attempt: for each package, sum up the values in the pile below that package.

Running time: $O(k^4)$.

(There are $O(k^2)$ packages and each of them has $O(k^2)$ packages below it.)

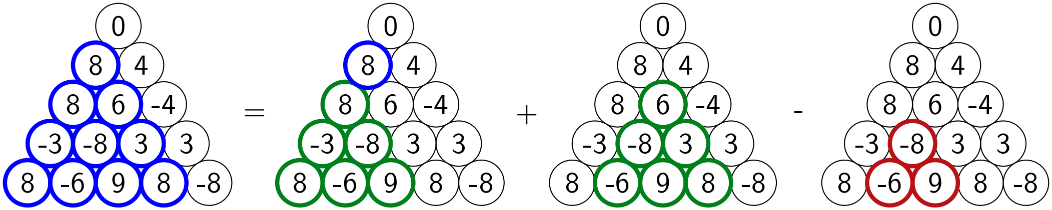
Alice's Accumulation – Partial Sums

Idea: precompute partial sums in each row (in time $O(k^2)$). Then we get a solution with running time $O(k^3)$:



Alice's Accumulation – Recursion

Idea: find a recursion.



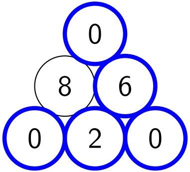
Alice's Accumulation – Dynamic Programming

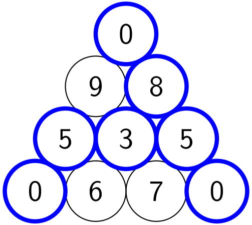
Hence, dynamic programming. Running time: $O(k^2)$.

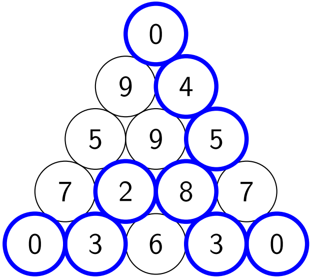
```
// Given: D[i][j] = value of the j-th ball in the i-th row
// Compute: P[i][j] = sum of sub-pile rooted at (i,j)

for (int i = k-1; i >= 0; --i) {
  for (int j = 0; j < i; ++j) {
    if (i == k-1)
      P[i][j] = D[i][j];
    else if (i == k-2)
      P[i][j] = D[i][j] + P[i+1][j] + P[i+1][j+1];
    else
      P[i][j] = D[i][j] + P[i+1][j] + P[i+1][j+1] - P[i+2][j+1];
  }
}
```

Bob's Burden – Understanding the Problem



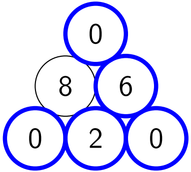


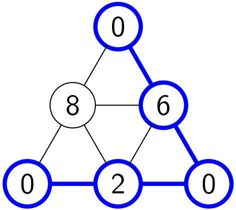


Bob's Burden – Modelling the Problem

Model: graph on B_{ij}, edge $\iff$ disks touch

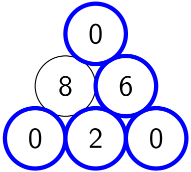
Bob's Burden – Modelling the Problem

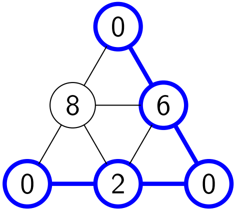




Model: graph on B_{ij}, edge $\iff$ disks touch
$\implies$ looking for a tree spanning the apices (not a spanning tree of the whole graph)

Bob's Burden – Modelling the Problem





Model: graph on B_{ij}, edge $\iff$ disks touch

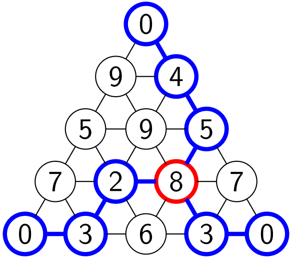
$\implies$ looking for a tree spanning the apices (not a spanning tree of the whole graph)

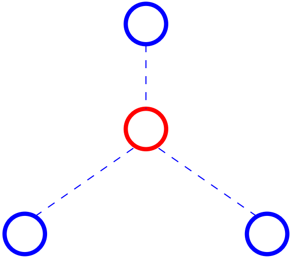
Q1: What does such a tree look like?

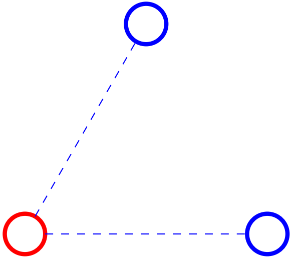
Q2: How to model the weights?

Q3: How to compute an optimum tree?

Bob's Burden – What does the Tree look like?





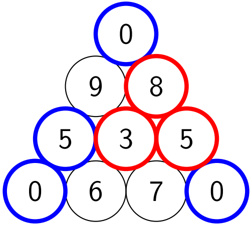


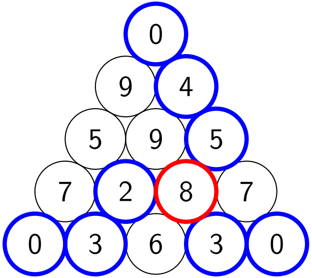
Obs. The tree consists of a center vertex c and optimum paths between c and the three apices.

Bob's Burden – Weights and Distances

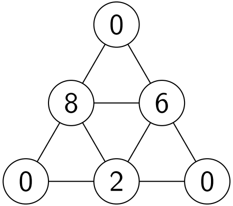
Task: Find center ball
with minimum
weight + distances.

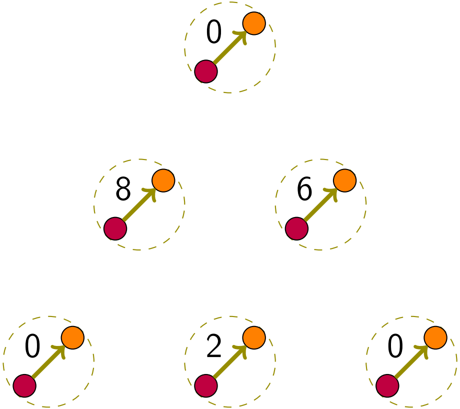
Weight: own weight of ball
Distances: to triangle apex
Center: may not be unique





Bob's Burden – Modelling Weights and Distances

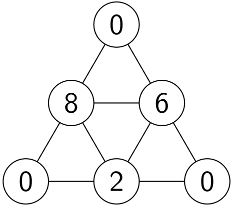


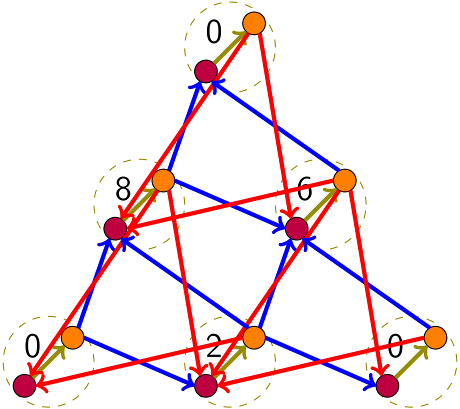


Graph model:

- B^{in} and B^{out} for each ball B,
- interior edge with the ball's weight,
- incoming/outgoing 0-edges to neighbors.

Bob's Burden – Modelling Weights and Distances





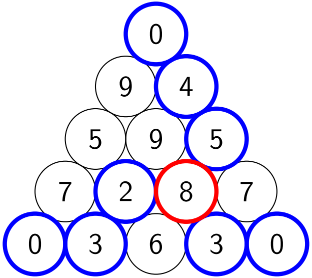
Graph model:

- B^{in} and B^{out} for each ball B,
- interior edge with the ball's weight,
- incoming/outgoing 0-edges to neighbors.

Bob's Burden – Computation 1st Subtask

First subtask: $k \leq 40 \;\Rightarrow\; \approx 800$ balls

For each candidate ball B,
we compute the distances to the apices:
Either by running Dijkstra from B^{out}.
We read the distances stored at
$B_{11}^{\text{out}}, B_{k1}^{\text{out}}, B_{kk}^{\text{out}}$.

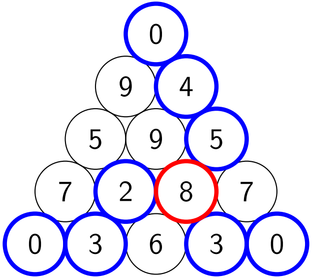


Bob's Burden – Computation 1st Subtask

First subtask: $k \leq 40 \;\Rightarrow\; \approx 800$ balls

For each candidate ball B,
we compute the distances to the apices:
Either by running Dijkstra from B^{out}.
We read the distances stored at
$B_{11}^{\mathrm{out}}, B_{k1}^{\mathrm{out}}, B_{kk}^{\mathrm{out}}$.
Runtime:

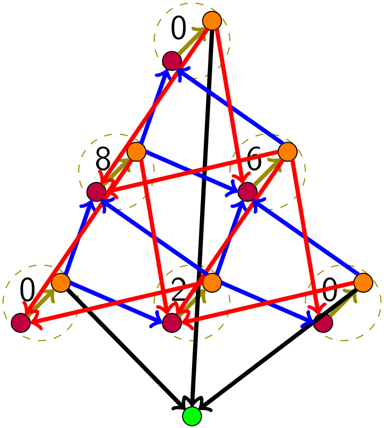
- $O(k^2)$ Dijkstra runs
- Runtime of Dijkstra:
  $O(n \log n + m) \to O(k^2 \log k)$
- Total running time: $O(k^4 \log k)$



Bob's Burden – Computation 1st Subtask

First subtask: $k \leq 40 \;\Rightarrow\; \approx 800$ balls

For each candidate ball B,
we compute the distances to the apices:
Or by Min Cost Max Flow (SSP version)
from B^{out} to a sink reachable from
$B^{\text{out}}_{11}, B^{\text{out}}_{k1}, B^{\text{out}}_{kk}$.
Sum of distances: `find_flow_cost(G)`.

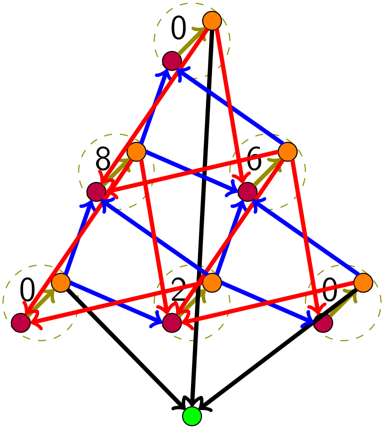


Bob's Burden – Computation 1st Subtask

First subtask: $k \leq 40 \;\Rightarrow\; \approx 800$ balls

For each candidate ball B,
we compute the distances to the apices:
Or by Min Cost Max Flow (SSP version)
from B^{out} to a sink reachable from
$B^{\mathrm{out}}_{11}, B^{\mathrm{out}}_{k1}, B^{\mathrm{out}}_{kk}$.
Sum of distances: `find_flow_cost(G)`.
Runtime:

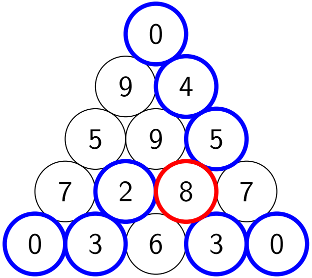
- $O(k^2)$ SSP Min Cost Max Flow runs
- Runtime of SSP:
  $O(|f|(m + n \log n) \rightarrow O(k^2 \log k)$
- Total running time: $O(k^4 \log k)$



Bob's Burden – Full Solution

Full solution: $k \leq 800 \;\Rightarrow\; \approx 320'000$ balls

We probably still have to look at each ball as a candidate.
How can we avoid the many (costly) Dijkstra runs?

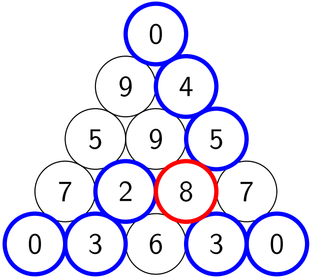


Bob's Burden – Full Solution

Full solution: $k \leq 800 \ \Rightarrow \ \approx 320'000$ balls

We probably still have to look at each ball as a candidate.
How can we avoid the many (costly) Dijkstra runs?

Do Dijkstra **only 3 times**, from B_{11}^{out}, B_{k1}^{out} and B_{kk}^{out}!



Bob's Burden – Full Solution

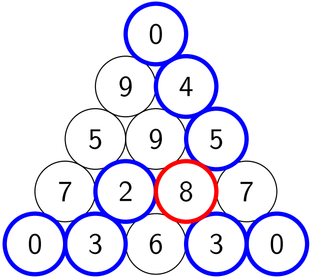
Full solution: $k \leq 800 \;\Rightarrow\; \approx 320'000$ balls

We probably still have to look at each ball as a candidate.
How can we avoid the many (costly) Dijkstra runs?

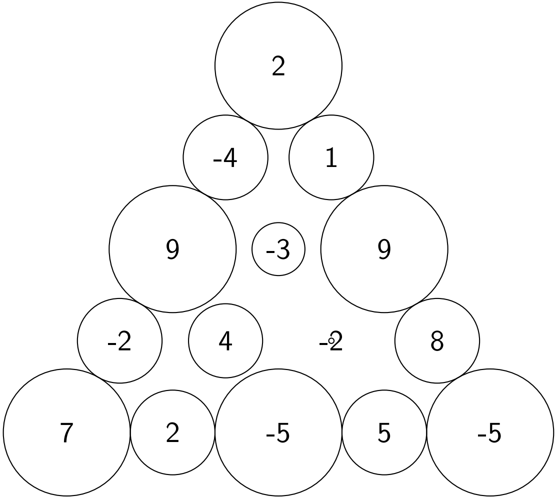
Do Dijkstra **only 3 times**, from B_{11}^{out}, B_{k1}^{out} and B_{kk}^{out}!
For each candidate ball B, sum up:

- the weight of B;
- the distances from B_{11}^{out}, B_{k1}^{out} and B_{kk}^{out} to B^{in}.

Runtime: $O(k^2 \log k)$.



Carol's Configuration – Problem Statement



Carol's Configuration – Boundary Only

This case can be solved "by hand".

Carol's Configuration – Boundary Only

This case can be solved "by hand".

Note that the balls on the boundary form a cyclic structure.

Carol's Configuration – Boundary Only

This case can be solved "by hand".

Note that the balls on the boundary form a cyclic structure.

If k is even, this cycle has odd length $(3k-3)$:

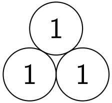
Carol's Configuration – Boundary Only

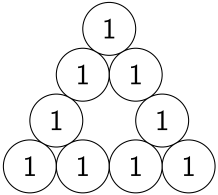
This case can be solved "by hand".

Note that the balls on the boundary form a cyclic structure.

If k is even, this cycle has odd length $(3k-3)$:

The cycle stays connected
if and only if all radii are 1.





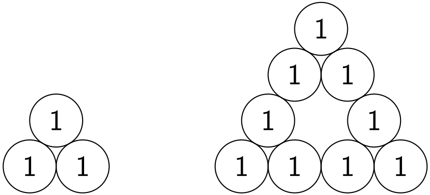
Carol's Configuration – Boundary Only

This case can be solved "by hand".

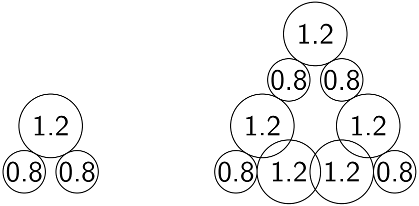
Note that the balls on the boundary form a cyclic structure.

If k is even, this cycle has odd length $(3k-3)$:

The cycle stays connected
if and only if all radii are 1.



If any radius is different from 1,
then it will be disconnected or intersecting.



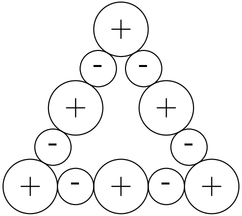
Carol's Configuration – Boundary Only

If k is odd, the cycle has even length $(3k-3)$:

Carol's Configuration – Boundary Only

If k is odd, the cycle has even length $(3k-3)$:

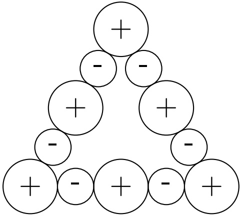
- Set the radius of B_{11} (and the other corners) to $1+x$, $0 \leq x \leq 1$. It cannot be smaller because the two neighbors would intersect otherwise.
- The radii on the cycle alternate between $1+x$ and $1-x$.



Carol's Configuration – Boundary Only

If k is odd, the cycle has even length $(3k-3)$:

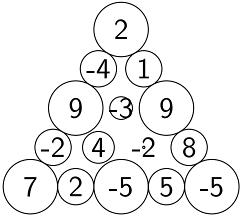
- Set the radius of B_{11} (and the other corners) to $1+x$, $0 \leq x \leq 1$. It cannot be smaller because the two neighbors would intersect otherwise.
- The radii on the cycle alternate between $1+x$ and $1-x$.



The objective is a linear function of the form $(1+x)c_1 + (1-x)c_2$.
Any monotone function on a closed interval I takes its maximum value on the boundary of I, i.e., at $x=0$ or $x=1$.

Carol's Configuration – Full Solution

For the general case we will formulate a Linear Program.



Carol's Configuration – Full Solution

For the general case we will formulate a Linear Program.

$$
\begin{array}{lll}
\text{variables:} & \{r_{ij}\} & \text{for each ball } B_{ij} \\
\text{maximize} & \sum r_{ij} v_{ij} & \\
\text{subject to} & r_{ij} + r_{i'j'} \leq 2, & \text{for all } B_{ij} \text{ and } B_{i'j'} \text{ that are } \textcolor{blue}{\text{direct neighbors}} \\
 & r_{ij} + r_{i'j'} \leq 2\sqrt{3}, & \text{for all } B_{ij} \text{ and } B_{i'j'} \text{ that are } \textcolor{green}{\text{indirect neighbors}} \\
 & r_{ij} + r_{i'j'} = 2, & \text{for all } B_{ij} \text{ and } B_{i'j'} \text{ that are direct neighbors on the boundary} \\
 & r_{ij} \geq 0, & \text{for each ball } B_{ij}
\end{array}
$$